TEMA 7

Contenido

Programación AJAX en javascript.

Caso práctico
*En estos últimos meses, **Antonio** ha realizado un montón de trabajos en el proyecto, y prácticamente ha terminado todas las tareas. **Juan** le dice que todo el trabajo realizado está muy bien, pero que tendría que actualizar algunos de los procesos para darle un toque de modernidad a la aplicación.*
*Entre las mejoras que le recomienda Juan, están la de utilizar efectos, más dinamismo, usar AJAX (JavaScript Asíncrono y XML) en las validaciones de los formularios, o en cierto tipo de consultas, etc. El término AJAX le suena muy complicado a **Antonio**, pero **Juan** lo convence rápidamente para que intente hacerlo, ya que no necesita aprender ningún lenguaje nuevo. Utilizando un nuevo objeto de JavaScript, con sus propiedades y métodos va a poder emplear AJAX en sus aplicaciones actuales.*
*Además, **Juan** lo anima a que se ponga a estudiar rápido el tema de AJAX, ya que al final, le va a dar una pequeña sorpresa, con una librería que le va a facilitar enormemente el programar con AJAX y conseguir dar buenos efectos y mayor dinamismo al proyecto web. Esa librería gratuita tiene el respaldo de grandes compañías a nivel mundial, que la están utilizando actualmente. También cuenta con infinidad de complementos, para que pueda modernizar su web todo lo que quiera, y todo ello programando muy pocas líneas de código y en un tiempo de desarrollo relativamente corto.*

1.- Introducción a AJAX.

Caso práctico
*AJAX es una tecnología crucial en lo que se conoce como web 2.0. A **Antonio** le atrae mucho el tema, ya que ha visto que con AJAX se pueden hacer envíos y consultas al servidor, sin tener que recargar las páginas web o cambiar de página, con lo que se consigue que sea todo más interactivo y adaptado a las nuevas tendencias. **Antonio** analiza la tecnología AJAX, sus orígenes, y el objeto que se utiliza para realizar las peticiones al servidor y gestionar las respuestas. Su directora **Ada**, le facilita unas direcciones muy interesantes con contenidos y ejemplos de algunas aplicaciones AJAX de antiguos proyectos realizados en la empresa.*

El término AJAX (JavaScript Asíncrono y XML) es una técnica de desarrollo web, que permite comunicar el navegador del usuario con el servidor, en un segundo plano. De esta forma, se podrían realizar peticiones al servidor sin tener que recargar la página, y podríamos gestionar esas respuestas, que nos permitirían actualizar los contenidos de nuestra página, sin tener que realizar recargas.

El término AJAX se presentó por primera vez en el artículo "A New Approach to Web Applications", publicado por Jesse James Carrett el 18 de febrero de 2005.

AJAX no es una tecnología nueva. Son realmente muchas tecnologías, cada una destacando por su propio mérito, pero que se unen con los siguientes objetivos:

- ✓ Conseguir una presentación basada en estándares, usando XHTML, CSS y un uso amplio de técnicas del DOM, para poder mostrar la información de forma dinámica e interactiva.
- ✓ Intercambio y manipulación de datos, usando XML y XSLT.
- ✓ Recuperación de datos de forma asíncrona, usando el objeto `XMLHTTPRequest`.
- ✓ Uso de JavaScript, para unir todos los componentes.

Las tecnologías que forman AJAX son:

- ✓ **XHTML** y **CSS**, para la presentación basada en estándares.
- ✓ **DOM**, para la interacción y manipulación dinámica de la presentación.
- ✓ **XML**, **XSLT** y **JSON**, para el intercambio y manipulación de información.
- ✓ **XMLHTTPRequest**, para el intercambio asíncrono de información.
- ✓ **JavaScript**, para unir todos los componentes anteriores.

El modelo clásico de aplicaciones Web funciona de la siguiente forma: la mayoría de las acciones del usuario se producen en la interfaz, disparando solicitudes HTTP al servidor web. El servidor efectúa un proceso (recopila información, realiza las acciones oportunas), y devuelve una página HTML al cliente. Este es un modelo adaptado del uso original de la Web como medio hipertextual, pero a nivel de aplicaciones de software, este tipo de modelo no es necesariamente el más recomendable.

Cada vez que se realiza una petición al servidor, el usuario lo único que puede hacer es esperar, ya que muchas veces la página cambia a otra diferente, y hasta que no reciba todos los datos del servidor, no se mostrará el resultado, con lo que el usuario no podrá interactuar de ninguna manera con el navegador. Con AJAX, lo que se intenta evitar, son esencialmente esas esperas. El cliente podrá hacer solicitudes al servidor, mientras el navegador sigue mostrando la misma página web, y cuando el navegador reciba una respuesta del servidor, la mostrará al cliente y todo ello sin recargar o cambiar de página.

AJAX es utilizado por muchas empresas y productos hoy en día. Por ejemplo, Google utiliza AJAX en aplicaciones como Gmail, Google Suggest, Google Maps.., así como Flickr, Amazon, etc.

Son muchas las razones para usar AJAX:
- ✓ Está basado en estándares abiertos.
- ✓ Su usabilidad.
- ✓ Válido en cualquier plataforma y navegador.
- ✓ Beneficios que aporta a las aplicaciones web.
- ✓ Compatible con Flash.
- ✓ Es la base de la web 2.0.
- ✓ Es independiente del tipo de tecnología de servidor utilizada.
- ✓ Mejora la estética de la web.

1.1.- Requerimientos previos.

A la hora de trabajar con AJAX debemos tener en cuenta una serie de requisitos previos, necesarios para la programación con esta metodología.

Hasta este momento, nuestras aplicaciones de JavaScript no necesitaban de un servidor web para funcionar, salvo en el caso de querer enviar los datos de un formulario y almacenarlos en una base de datos. Es más, todas las aplicaciones de JavaScript que has realizado, las has probado directamente abriéndolas con el navegador o haciendo doble click sobre el fichero .HTML.

Para la programación con AJAX vamos a necesitar de un servidor web, ya que las peticiones AJAX que hagamos, las haremos a un servidor. Los componentes que necesitamos son:
- ✓ Servidor web (apache, ligHTTPd, IIS, etc).
- ✓ Servidor de bases de datos (MySQL, Postgresql, etc).
- ✓ Lenguaje de servidor (PHP, ASP, etc).

Podríamos instalar cada uno de esos componentes por separado, pero muchas veces lo más cómodo es instalar alguna aplicación que los agrupe a todos sin instalarlos de forma individual. Hay varios tipos de aplicaciones de ese tipo, que se pueden categorizar en dos, diferenciadas por el tipo de sistema operativo sobre el que funcionan:
- ✓ servidor LAMP (Linux, Apache, MySQL y PHP).
- ✓ servidor WAMP (Windows, Apache, MySQL y PHP).

Una aplicación de este tipo, muy utilizada, puede ser XAMPP (tanto para Windows, como para Linux).

Esta aplicación podrás instalarla incluso en una memoria USB y ejecutarla en cualquier ordenador, con lo que tendrás siempre disponible un servidor web, para programar tus aplicaciones AJAX.

Servidor XAMPP (Apache, MySQL, PHP). http://www.apachefriends.org/es/xampp.html
Cómo intalar XAMPP http://www.youtube.com/watch?feature=player_embedded&v=LHomeNG8Iz0

Un complemento muy recomendable para la programación con AJAX, es la extensión gratuita **Firebug** de Firefox. Esta extensión es muy útil, ya que con ella podremos detectar errores en las peticiones AJAX, ver los datos que enviamos, en que formato van, qué resultado obtenemos en la petición y un montón de posibilidades más, como inspeccionar todo el DOM de nuestro documento, las hojas de estilo, detectar errores en la programación con JavaScript, etc.

Complemento Firebug para Firefox. http://getfirebug.com/

1.2.- Comunicación asíncrona.

Como ya te comentábamos en la introducción a AJAX, la mayoría de las aplicaciones web funcionan de la siguiente forma:

1. El usuario solicita algo al servidor.
2. El servidor ejecuta los procesos solicitados (búsqueda de información, consulta a una base de datos, lectura de fichero, cálculos numéricos, etc.).
3. Cuando el servidor termina, devuelve los resultados al cliente.

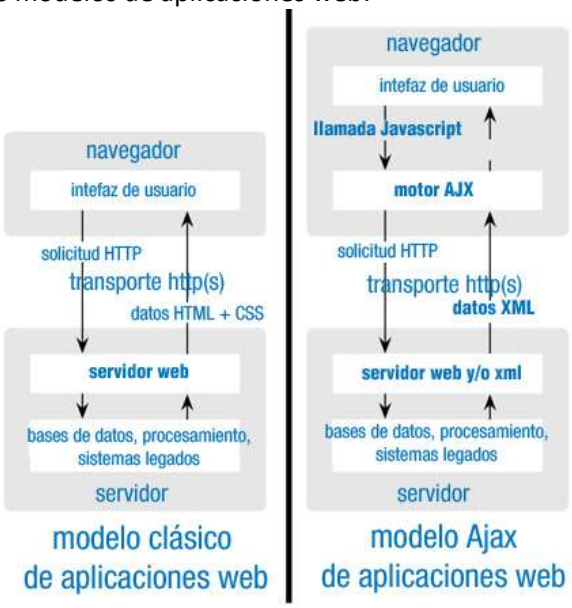
En el paso 2, mientras se ejecutan los procesos en el servidor, el cliente lo único que puede hacer es esperar, ya que el navegador está bloqueado en espera de recibir la información con los resultados del servidor.

Una aplicación AJAX, cambia la metodología de funcionamiento de una aplicación web, en el sentido de que, elimina las esperas y los bloqueos que se producen en el cliente. Es decir, el usuario podrá seguir interactuando con la página web, mientras se realiza la petición al servidor. En el momento de tener una respuesta confirmada del servidor, ésta será mostrada al cliente, o bien se ejecutarán las acciones que el programador de la página web haya definido.

Mira el siguiente gráfico, en el que se comparan los dos modelos de aplicaciones web:

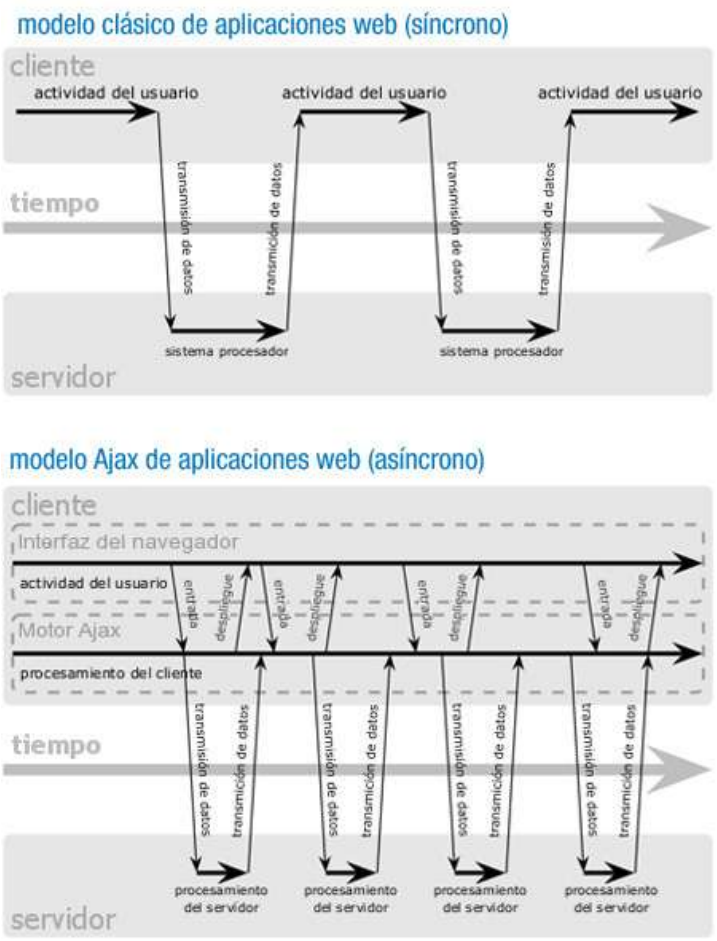
¿Cómo se consigue realizar la petición al servidor sin bloquear el navegador?

Para poder realizar las peticiones al servidor sin que el navegador se quede bloqueado, tendremos que hacer uso del motor AJAX (programado en JavaScript y que generalmente se encuentra en un frame oculto). Este motor se encarga de gestionar las peticiones AJAX del usuario, y de comunicarse con el servidor. Es justamente este motor, el que permite que la interacción suceda de forma asíncrona (independientemente de la comunicación con el servidor). Así, de esta forma, el usuario no tendrá que estar pendiente del icono de indicador de carga del navegador, o viendo una pantalla en blanco.



Cada acción del usuario, que normalmente generaría una petición HTTP al servidor, se va a convertir en una petición AJAX con esa solicitud, y será este motor, el que se encargará de todo el proceso de comunicación y obtención de datos de forma asíncrona con el servidor, y todo ello sin frenar la interacción del usuario con la aplicación.

"Hablar, es el arte de sofocar e interrumpir el pensamiento."
Carlyle, Thomas



¿Según los gráficos anteriores, en qué modelo de aplicación web la actividad del usuario se ve interrumpida o bloqueada por la espera de las respuestas del servidor?

- (•) **Clásico.**
- ( ) AJAX.

En este modelo cuando hacemos una petición al servidor, tendremos que esperar la respuesta del mismo, y mientras esperamos, la página estará bloqueada y no nos dejará hacer ninguna otra cosa.

1.3.- El API XMLHttpRequest.

El corazón de AJAX es una API denominada `XMLHTTPRequest` (XHR), disponible en los lenguajes de scripting en el lado del cliente, tales como JavaScript. Se utiliza para realizar peticiones, HTTP o HTTPS, directamente al servidor web, y para cargar las respuestas directamente en la página del cliente. Los datos que recibamos desde el servidor se podrán recibir en forma de texto plano o texto XML. Estos datos, podrán ser utilizados para modificar el DOM del documento actual, sin tener que recargar la página, o también podrán ser evaluados con JavaScript, si son recibidos en formato JSON.
`XMLHTTPRequest` juega un papel muy importante en la técnica AJAX, ya que sin este objeto, no sería posible realizar las peticiones asíncronas al servidor.

El concepto que está detrás del objeto `XMLHTTPRequest`, surgió gracias a los desarrolladores de Outlook Web Access (de Microsoft), en su desarrollo de Microsoft Exchange Server 2000. La interfaz `IXMLHTTPRequest`, se desarrolló e implementó en la segunda versión de la librería MSXML, empleando este concepto. Con el navegador Internet Explorer 5.0 en Marzo de 1999, se permitió el acceso a dicha interfaz a través de ActiveX.

Posteriormente la fundación Mozilla, desarrolló e implementó una interfaz llamada `nslXMLHTTPRequest`, dentro de su motor Gecko. Esa interfaz, se desarrolló adaptándose lo más posible

a la interfaz implementada por Microsoft. Mozilla creó un envoltorio para usar esa interfaz, a través de un objeto JavaScript, el cuál denominó `XMLHTTPRequest`. El objeto `XMLHTTPRequest` fue accesible en la versión 0.6 de Gecko, en diciembre de 2000, pero no fue completamente funcional, hasta Junio de 2002 con la versión 1.0 de Gecko. El objeto `XMLHTTPRequest`, se convirtió de hecho en un estándar entre múltiples navegadores, como Safari 1.2, Konqueror, Opera 8.0 e iCab 3.0b352 en el año 2005.

El W3C publicó una especificación-borrador para el objeto `XMLHTTPRequest`, el 5 de Abril de 2006. Su objetivo era crear un documento con las especificaciones mínimas de interoperabilidad, basadas en las diferentes implementaciones que había hasta ese momento. La última revisión de este objeto, se realizó en Noviembre de 2009.

Microsoft añadió el objeto `XMLHTTPRequest` a su lenguaje de script, con la versión de Internet Explorer 7.0 en Octubre de 2006.

Con la llegada de las librerías cross-browser (*capacidad que una web, aplicación web, construcciónHTML o script del lado del cliente tiene y que permite que sea soportada por todos los navegadores, es decir que se pueda mostrar o ejecutar de forma correcta en cualquier navegador*) como jQuery, Prototype, etc, los programadores pueden utilizar toda la funcionalidad de `XMLHTTPRequest`, sin codificar directamente sobre la API, con lo que se acelera muchísimo el desarrollo de aplicaciones AJAX.

En febrero de 2008, la W3C publicó otro borrador denominado "`XMLHTTPRequest Nivel 2`". Este nivel consiste en extender la funcionalidad del objeto `XMLHTTPRequest`, incluyendo, pero no limitando, el soporte para peticiones cross-site (*peticiones situadas en diferentes dominios*), gestión de byte streams (*agrupación de bits en unidades que darán lugar a los bytes*), progreso de eventos, etc. Esta última revisión de la especificación, sigue estando en estado "working draft" (borrador), a septiembre de 2010.

Una de las limitaciones de `XMLHTTPRequest` es que, por seguridad, sólo nos deja realizar peticiones AJAX, a las páginas que se encuentren hospedadas en el mismo DOMinio, desde el cual se está realizando la petición AJAX.

"Tan sólo por la educación puede el hombre llegar a ser hombre. El hombre no es más que lo que la educación hace de él."

Kant, Immanuel

1.3.1.- Creación del objeto XMLHttpRequest.

Para poder programar con AJAX, necesitamos crear un objeto del tipo `XMLHTTPRequest`, que va a ser el que nos permitirá realizar las peticiones en segundo plano al servidor web.

Una vez más, nos vamos a encontrar con el problema de Internet Explorer, que, dependiendo de la versión que utilicemos, tendremos que crear el objeto de una manera o de otra. Aquí tienes un ejemplo de una función cross-browser, que devuelve un objeto del tipo XHR (`XMLHTTPRequest`):

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHTTPRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHTTPRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHTTPRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('MsXML2.XMLHTTP.5.0', 'MsXML2.XMLHTTP.4.0',
      'MsXML2.XMLHTTP.3.0', 'MsXML2.XMLHTTP', 'Microsoft.XMLHTTP');
      for (var i = 0; i < versionesIE.length; i++){
         try{
            /* Se intenta crear el objeto en Internet Explorer comenzando
            en la versión más moderna del objeto hasta la primera versión.
            En el momento que se consiga crear el objeto, saldrá del bucle
```

```
            devolviendo el nuevo objeto creado. */
            return new ActiveXObject(versionesIE[i]);
         } catch (errorControlado) {}//Capturamos el error,
      }
   }

   /* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

     Más información sobre gestión de errores en:
     HTTP://www.javascriptkit.com/javatutors/trycatch2.sHTML
   */

   throw new Error("No se pudo crear el objeto XMLHTTPRequest");
}

// para crear un objeto XHR lo podremos hacer con la siguiente llamada.
var objetoAJAX = new objetoXHR();
```

Una opción muy interesante, consiste en hacer una librería llamada, por ejemplo, `funciones.js`, que contenga el código de tus funciones más interesantes como, `crearEvento()`, `objetoXHR()`, etc. De esta manera, irás creando tus propios recursos, con el código de JavaScript que más uses en tus aplicaciones.

"El éxito no se logra sólo con cualidades especiales, es sobre todo un trabajo de constancia, de método y de organización."

Sergent, J.P.

1.3.2.- Métodos del objeto XMLHttpRequest.

El objeto `XMLHTTPRequest` dispone de los siguientes métodos, que nos permitirán realizar peticiones asíncronas al servidor:

Metodo	Descripción
`abort()`	Cancela la solicitud actual.
`getAllResponseHeaders()`	Devuelve la información completa de la cabecera.
`getResponseHeader()`	Devuelve la información específica de la cabecera.
`open(metodo, url, async, usuario, password)`	Especifica el tipo de solicitud, la URL, si la solicitud se debe gestionar de forma asíncrona o no, y otros atributos opcionales de la solicitud. ✓ **método**: indicamos el tipo de solicitud: `GET` o `POST`. ✓ **url**: la dirección del fichero al que le enviamos las peticiones en el servidor. ✓ **async**: `true` (asíncrona) o `false` (síncrona). ✓ **usuario y password**: si fuese necesaria la autenticación en él
`send (datos)`	Envía la solicitud al servidor. ✓ datos: Se usa en el caso de que estemos utilizando el método `POST`, como método de envío. Si usamos `GET`, datos será `null`.
`setRequestHeader()`	Añade el par etiqueta/valor a la cabecera de datos que se enviará al servidor.

Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes copiar los ficheros del ejemplo, dentro de la raíz del servidor web, en una carpeta a la que llamaremos, por ejemplo, `web/dwec07132`. Arrancaremos el servidor web e iremos a la dirección `HTTP://localhost/web/dwec07132`

Puedes utilizar Firebug, para comprobar cómo se realiza la petición AJAX.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 1.3.2 - AJAX SINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud SINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/>
      Contenedor resultados:<div id="resultados"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
//////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos en el objeto con ID resultados el contenido
   // del fichero datos.txt empleando una petición AJAX.
   cargarSync(document.getElementById("resultados"),"fecha.php");
}

////////////////////////////////////////////////////////
// Función cargarSync: carga el contenido de la url
// en el objeto que se le pasa como referencia,
// usando una petición AJAX de forma SINCRONA.
////////////////////////////////////////////////////////
function cargarSync(objeto, url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, false); //Abrimos la url, false=SINCRONA

      // Hacemos la petición al servidor. Como parámetro del método send:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

fecha.php

```
<?php
   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try {
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */
   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function()  {
   function w3c_crearEvento(elemento, evento, mifuncion)    {
      elemento.addEventListener(evento, mifuncion, false);
   }
   function ie_crearEvento(elemento, evento, mifuncion)            {
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }
   if (typeof window.addEventListener !== 'undefined')      {
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function cargarSync(objeto, url) {
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, false); //Abrimos la url, false=SINCRONA

      // Hacemos la petición al servidor. Como parámetro del método send:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

En esta función se realiza una petición AJAX, pero de forma síncrona (comportamiento normal del navegador). En dicha petición se realiza la carga del fichero indicado en la url (debe ser un fichero perteneciente al mismo DOMinio del servidor). La respuesta (**`responseText`**), que obtenemos en esa petición, se coloca en un DIV, con la función personalizada `textoDIV` (su código fuente está en el fichero `funciones.js`).

1.3.3.- Propiedades del objeto XMLHttpRequest.

El objeto **`XMLHTTPRequest`**, dispone de las siguientes propiedades, que nos facilitan información sobre el estado de la petición al servidor, y donde recibiremos los datos de la respuesta devuelta en la petición AJAX:

Propiedad	Descripción
`onreadystatechange`	Almacena una función (o el nombre de una función), que será llamada automáticamente, cada vez que se produzca un cambio en la propiedad `readyState`.
`readyState`	Almacena el estado de la petición `XMLHTTPRequest`. Posibles estados, del 0 al 4: ✓ **0**: solicitud no inicializada. ✓ **1**: conexión establecida con el servidor. ✓ **2**: solicitud recibida. ✓ **3**: procesando solicitud. ✓ **4**: solicitud ya terminada y la respuesta está disponible.
`responseText`	Contiene los datos de respuesta, como una cadena de texto.
`responseXML`	Contiene los datos de respuesta, en formato XML.
`status`	Contiene el estado numérico, devuelto en la petición al servidor (por ejemplo: "`404`" para "No encontrado" o "`200`" para "OK").
`statusText`	Contiene el estado en formato texto, devuelto en la petición al servidor (por ejemplo: "`Not Found`" o "`OK`").

Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes copiar los siguientes ficheros dentro de la raíz del servidor web, en la carpeta `web/dwes07133`, arrancar el servidor web e ir a la dirección **`HTTP://localhost/web/dwec07133`**

Puedes utilizar Firebug, para comprobar cómo se está realizando la petición AJAX.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 1.3.3 - AJAX ASINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/>
      Contenedor resultados:<div id="resultados"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
//////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos en el objeto con ID resultados el contenido
   // del fichero datos.txt empleando una petición AJAX.
   cargarAsync(document.getElementById("resultados"),"fecha.php");
}

//////////////////////////////////////////////////////
// Función cargarSync: carga el contenido de la url
// en el objeto que se le pasa como referencia,
// usando una petición AJAX de forma ASINCRONA.
//////////////////////////////////////////////////////
function cargarAsync(objeto, url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // Hacemos la petición al servidor. Como parámetro:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

fecha.php

```
<?php
   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
//////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
//////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest) {
      // El navegador implementa la interfaz XHR de forma nativa
```

```
      return new XMLHttpRequest();
   }else if (window.ActiveXObject)   {
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++)  {
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}

////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion)   {
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion)    {
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined')     {
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined')    {
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function cargarAsync(objeto, url) {
   if (miXHR){
```

```
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // Hacemos la petición al servidor. Como parámetro:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

En esta función se realiza una petición AJAX, pero de forma asíncrona. En dicha petición se realiza la carga del fichero indicado en la url (debe ser un fichero perteneciente al mismo DOMinio del servidor). La respuesta (**`responseText`**), que obtenemos en esa petición, se coloca en un DIV, con la función personalizada `textoDIV` (su código fuente está en el fichero `funciones.js`). Si ejecutas el ejemplo anterior, verás que no se muestra nada, ya que no hemos gestionado correctamente la respuesta recibida de forma asíncrona. En el siguiente apartado 2, de esta unidad, veremos cómo corregir ese fallo y realizar correctamente esa operación.

2.- Envío y recepción de datos de forma asíncrona.

Caso práctico
*Ahora que **Antonio** ya conoce el objeto `XMLHTTPRequest`, con sus propiedades y métodos, se centra en cómo se realiza la petición al servidor de forma asíncrona, y cómo se gestionan los estados y las respuestas que nos devuelve. También estudia qué formatos tiene para enviar datos al servidor, y en qué formatos va a recibir esos resultados.*
*De entre todos los formatos de recepción de datos, **Juan** recomienda a **Antonio** uno de ellos: el formato JSON. Dicho formato, utiliza la nomenclatura de JavaScript, para enviar los resultados. De esta forma puede utilizar dichos resultados directamente en la aplicación JavaScript, sin tener que realizar prácticamente ningún tipo de conversiones intermedias.*

En el apartado 1.3.3., hemos visto un ejemplo de una petición asíncrona al servidor, pero vimos que éramos incapaces de mostrar correctamente el resultado en el contenedor `resultados`.

```
function cargarAsync(objeto, url) {
   if (miXHR) {
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // Hacemos la petición al servidor. Como parámetro:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

Si ejecutas el ejemplo del apartado 1.3.3., verás que no se muestra nada en el contenedor resultados, y la razón es porque, cuando accedemos a la propiedad `miXHR.responseText`, ésta no contiene nada. Eso es debido a la solicitud asíncrona. Si recuerdas, cuando hicimos el ejemplo con una solicitud síncrona, se mostró la alerta de "***Comenzamos la petición AJAX***", aceptaste el mensaje, y justo 2 segundos después recibimos la alerta de "***Terminó la petición AJAX***". En el modo **síncrono**, el navegador cuando hace la petición al servidor, con el método `miXHR.send()`, se queda esperando hasta que termine la solicitud (y por lo tanto nosotros no podemos hacer otra cosa más que esperar ya que el navegador está bloqueado). Cuanto termina la solicitud, pasa a la siguiente línea: `textoDIV(objeto,....)`, y por tanto ya puede mostrar el contenido de `responseText`.

En el modo asíncrono, cuando aceptamos la primera alerta, prácticamente al instante se nos muestra la siguiente alerta. Esto es así, por que el navegador en una petición asíncrona, no espera a que termine esa solicitud, y continúa ejecutando las siguientes instrucciones. Por eso, cuando se hace la llamada con el método `miXHR.send()`, dentro de la función `cargarAsync()`, el navegador sigue ejecutando las dos instrucciones siguientes, sin esperar a que termine la solicitud al servidor. Y es por eso que no se muestra nada, ya que la propiedad `responseText`, no contiene ningún resultado todavía, puesto que la petición al servidor está todavía en ejecución. **¿Cuál será entonces la solución?**

Una primera solución que se nos podría ocurrir, sería la de poner un tiempo de espera o retardo, antes de ejecutar la instrucción `textoDIV`. Esto, lo único que va a hacer, es bloquear todavía más nuestro navegador, y además, tampoco sabemos exactamente lo que va a tardar el servidor web en procesar nuestra solicitud.

La segunda solución, consistiría en detectar cuándo se ha terminado la petición AJAX, y es entonces en ese momento, cuando accederemos a la propiedad `responseText` para coger los resultados. Ésta será la solución, que adoptaremos y que veremos en el apartado 2.1 de esta unidad.

2.1.- Estados de una solicitud asíncrona (parte I).

Cuando se realiza una petición asíncrona, dicha petición va a pasar por diferentes estados (del 0 al 4 - propiedad `readyState` del objeto XHR), independientemente de si el fichero solicitado al servidor, se encuentre o no.

Atención: dependiendo del navegador utilizado, habrá algunos estados que no son devueltos.

Cuando dicha petición termina, tendremos que comprobar cómo lo hizo, y para ello evaluamos la propiedad `status` que contiene el estado devuelto por el servidor: `200`: ***OK***, `404`: ***Not Found***, etc. Si `status` fue **OK** ya podremos comprobar, en la propiedad `responseText` o `responseXML` del objeto XHR, los datos devueltos por la petición.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.1 - AJAX ASINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/><br/>
      Contenedor resultados:<div id="resultados"></div>
      Estado de las solicitudes:
      <div id="estado"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos el fichero fecha.php de forma asíncrona.
   cargarAsync("fecha.php");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
```

```
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   switch(this.readyState){
      // Evaluamos el estado de la petición AJAX
      // Vamos mostrando el valor actual de readyState en cada llamada.
      case 0:  document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
         break;
      case 1: document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
         break;
      case 2: document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
         break;
      case 3: document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
         break;
      case 4: document.getElementById('estado').innerHTML += "4 - Completado.";
         if (this.status == 200){
            // Si el servidor ha devuelto un OK
            // Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
            textoDIV(document.getElementById("resultados"), this.responseText);
            alert("Terminó la petición AJAX");
         }
         break;
   }
}
```

fecha.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest) {
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject)   {
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++)  {
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.
```

```
      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}

//////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
//////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion)   {
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion)    {
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined')     {
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined')    {
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


//////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
//////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
//////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url usando una petición AJAX de forma
// ASINCRONA.
//////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

//////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
//////////////////////////////////////////////////////////
```

```
function estadoPeticion(){
   switch(this.readyState){
      // Evaluamos el estado de la petición AJAX
      // Vamos mostrando el valor actual de readyState en cada llamada.
      case 0:  document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
         break;
      case 1: document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
         break;
      case 2: document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
         break;
      case 3: document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
         break;
      case 4: document.getElementById('estado').innerHTML += "4 - Completado.";
         if (this.status == 200){
            // Si el servidor ha devuelto un OK
            // Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
            textoDIV(document.getElementById("resultados"), this.responseText);
            alert("Terminó la petición AJAX");
         }
         break;
   }
}
```

2.2.- Estados de una solicitud asíncrona (parte II).

El código del apartado 2.1, fue programado con fines didácticos para mostrar los 4 estados de la petición. El ejemplo completo, con una imagen animada indicadora de la actividad AJAX, se muestra a continuación:

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.2 - AJAX ASINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/><br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos el fichero fecha.php de forma asíncrona.
   cargarAsync("fecha.php");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
```

```
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 )  {
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

fecha.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
         /*
            Se intenta crear el objeto en Internet Explorer comenzando
            en la versión más moderna del objeto hasta la primera versión.
            En el momento que se consiga crear el objeto, saldrá del bucle
            devolviendo el nuevo objeto creado.
         */
         return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
```

```
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


//////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
//////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }

   else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


//////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
//////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

ajax-loader.gif

```
//////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
//////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="<img src='AJAX-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
```

```
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

2.3.- Envío de datos usando método GET.

Vamos a ver un ejemplo, en el que se realiza una petición AJAX a la página `procesar.php`, pasando dos parámetros: nombre y apellidos, usando el método **`GET`**.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.3 - AJAX ASINCRONO GET - POST</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php con los parámetros indicados en la URL
   cargarAsync("procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
```

```
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

procesar.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // Imprimimos un mensaje con los textos recibidos
   if (isset($_GET['nombre']))
      echo "Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}. ";
   else if (isset($_POST['nombre']))
      echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s");
?>
```

funciones.js

```
////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
   Emitimos un mensaje de error usando el objeto Error.

   Más información sobre gestión de errores en:
   http://www.javascriptkit.com/javatutors/trycatch2.shtml */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}
```

```
//////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
//////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


//////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
//////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php con los parámetros indicados en la URL
   cargarAsync("procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro");
}

//////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
//////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador AJAX antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='AJAX-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}
//////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
```

```
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

En la petición `GET`, los parámetros que pasemos en la solicitud, se enviarán formando parte de la URL. Por ejemplo: `procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro`. Cuando realizamos la petición por el método `GET`, te recordamos una vez más, que pondremos `null` como parámetro del método `send`, ya que los datos son enviados a la página procesar.php, formando parte de la URL: `send(null)`.

2.4.- Envío de datos usando método POST.

Vamos a ver un ejemplo en el que se realiza una petición AJAX, a la página procesar.php pasando dos párametros: nombre y apellidos, usando el método `POST`.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.3 - AJAX ASINCRONO GET - POST</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php
   // En este caso sólo ponemos los parámetros que pasaremos a la página procesar.php
   cargarAsync("nombre=Teresa&apellidos=Blanco Ferreiro");
}

////////////////////////////////////////////////////////
// Función cargarASync: carga el contenido con los parametros
// que se le vana a pasar a la petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(parametros){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
```

```
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      // Abrimos la conexión al servidor usando método POST y a la página procesar.php
      miXHR.open('POST', "procesar.php", true);   // Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // En las peticiones POST tenemos que enviar en la cabecera el Content-Type
      //ya que los datos se envían formando parte de la cabecera.
      miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

      // Hacemos la petición al servidor con los parámetros: nombre=Teresa&apellidos=Blanco...
      miXHR.send(parametros);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

procesar.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // Imprimimos un mensaje con los textos recibidos
   if (isset($_GET['nombre']))
      echo "Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}. ";
   else if (isset($_POST['nombre']))
      echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }
```

```
   /* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
   Emitimos un mensaje de error usando el objeto Error.

   Más información sobre gestión de errores en:
   http://www.javascriptkit.com/javatutors/trycatch2.shtml */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php
   // En este caso sólo ponemos los parámetros que pasaremos a la página procesar.php
   cargarAsync("nombre=Teresa&apellidos=Blanco Ferreiro");
}

////////////////////////////////////////////////////////
// Función cargarASync: carga el contenido con los parametros
// que se le vana a pasar a la petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(parametros){
   if (miXHR){
      // Activamos el indicador AJAX antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='AJAX-loader.gif'/>";

      // Abrimos la conexión al servidor usando método POST y a la página procesar.php
```

```
      miXHR.open('POST', "procesar.php", true);   // Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // En las peticiones POST tenemos que enviar en la cabecera el Content-Type
      //ya que los datos se envían formando parte de la cabecera.
      miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

      // Hacemos la petición al servidor con los parámetros: nombre=Teresa&apellidos=Blanco...
      miXHR.send(parametros);
   }
}
```

En este ejemplo, tuvimos que realizar los siguientes cambios para adaptarlo al método **POST**:

- ✓ La función `cargarAsync()`, recibirá los parámetros por `POST` en lugar de `GET`.
- ✓ El método `.open` se modifica por:

```
miXHR.open('POST', "procesar.php", true);
```

- ✓ Tenemos que crear una cabecera con el tipo de contenido que vamos a enviar, justo antes de enviar la petición con el método `.send()`:

```
miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

- ✓ En el método `.send()` escribiremos los parámetros (`nombre=Teresa&apellidos=Blanco Ferreiro`) que serán enviados por el método `POST`:

```
miXHR.send(parametros.
```

2.5.- Recepción de datos en formato XML.

Cuando realizamos una petición AJAX, que nos devuelve las respuestas en formato XML, dichos datos los tendremos que consultar en la propiedad **responseXML** del objeto XHR.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.3 - AJAX ASINCRONO GET - POST</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos los datos XML de forma asíncrona.
   // En este caso ponemos una página PHP que nos devuelve datos en formato XML
   // pero podríamos poner también el nombre de un fichero XML directamente: catalogos.xml
   // Se adjunta ejemplo de fichero XML.
   cargarAsync("datosxml.php");
}

////////////////////////////////////////////////////////
```

```
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada a cada cambio de estado de la petición AJAX
// cuando la respuesta del servidor es 200(fichero encontrado) y el estado de
// la solicitud es 4, accedemos a la propiedad responseText
////////////////////////////////////////////////////////
function estadoPeticion(){
   if (this.readyState==4 && this.status == 200)    {
      // Almacenamos el fichero XML en la variable datos.
      var datos=this.responseXML;

      // Tenemos que recorrer el fichero XML empleando los métodos del DOM
      // Array que contiene todos los CD's del fichero XML
      CDs= datos.documentElement.getElementsByTagName("CD");

      // En la variable salida compondremos el código HTML de la tabla a imprimir.
      salida="<table border='1'><tr><th>Titulo</th><th>Artista</th><th>Año</th></tr>";

      // Hacemos un bucle para recorrer todos los elementos CD.
      for (i=0;i<CDs.length;i++){
         salida+="<tr>";

         // Para cada CD leemos el título
         titulos=CDs[i].getElementsByTagName("TITLE");

         try{
            // Intentamos imprimir el contenido de ese elemento
            salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
         }catch (er){
            // En el caso de que no tenga contenido ese elemento, imprimimos un espacio en
blanco
            salida+= "<td> </td>";
         }

         // Para cada CD leemos el Artista
         titulos=CDs[i].getElementsByTagName("ARTIST");
         try{
            // Intentamos imprimir el contenido de ese elemento
            salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
         }catch (er){
            // En el caso de que no tenga contenido ese elemento, imprimimos un espacio en
blanco
            salida+="<td> </td>";
         }

         // Para cada CD leemos el Año
         titulos=CDs[i].getElementsByTagName("YEAR");
         try{
            // Intentamos imprimir el contenido de ese elemento
            salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
         }catch (er){// En el caso de que no tenga contenido ese elemento, imprimimos un
espacio en blanco
            salida+="<td> </td>";
         }

         // Podríamos seguir sacando más campos del fichero XML..

         // Cerramos la fila.
         salida+="</tr>";
```

```
        }

        // Cuando ya no hay más Cd's cerramos la tabla.
        salida+="</table>";

        // Desactivamos el indicador AJAX cuando termina la petición
        document.getElementById("indicador").innerHTML="";

        // Imprimimos la tabla dentro del contenedor resultados.
        document.getElementById("resultados").innerHTML=salida;
    }
}
```

datosxml.php

```
<?php
    // Para que el navegador no haga cache de los datos devueltos por la página PHP.
    header('Cache-Control: no-cache, must-revalidate');
    header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

    // Leemos el contenido del fichero XML
    // e imprimimos su contenido.
    // Muy importante indicar al navegador que va a recibir contenido XML
    // eso lo hacemos con la siguiente cabecera:
    header("Content-Type: text/xml");

    $ficheroxml="<?xml version=\"1.0\" encoding=\"utf-8\"?>";
    $ficheroxml.="
    <CATALOG>
    <CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>Hide your heart</TITLE>
        <ARTIST>Bonnie Tyler</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS Records</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1988</YEAR>
    </CD>
    <CD>
        <TITLE>Greatest Hits</TITLE>
        <ARTIST>Dolly Parton</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>RCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1982</YEAR>
    </CD>
    <CD>
        <TITLE>Still got the blues</TITLE>
        <ARTIST>Gary Moore</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Virgin records</COMPANY>
        <PRICE>10.20</PRICE>
        <YEAR>1990</YEAR>
    </CD>
    <CD>
        <TITLE>Eros</TITLE>
        <ARTIST>Eros Ramazzotti</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>BMG</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1997</YEAR>
    </CD>
    <CD>
        <TITLE>One night only</TITLE>
        <ARTIST>Bee Gees</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Polydor</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1998</YEAR>
    </CD>
    <CD>
```

```
      <TITLE>Sylvias Mother</TITLE>
      <ARTIST></ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS</COMPANY>
      <PRICE>8.10</PRICE>
      <YEAR>1973</YEAR>
   </CD>
   <CD>
      <TITLE>Maggie May</TITLE>
      <ARTIST>Rod Stewart</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Pickwick</COMPANY>
      <PRICE>8.50</PRICE>
      <YEAR>1990</YEAR>
   </CD>
   <CD>
      <TITLE>Romanza</TITLE>
      <ARTIST>Andrea Bocelli</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>10.80</PRICE>
      <YEAR>1996</YEAR>
   </CD>
   <CD>
      <TITLE>When a man loves a woman</TITLE>
      <ARTIST>Percy Sledge</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>8.70</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Black angel</TITLE>
      <ARTIST>Savage Rose</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Mega</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR></YEAR>
   </CD>
   <CD>
      <TITLE>1999 Grammy Nominees</TITLE>
      <ARTIST>Many</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Grammy</COMPANY>
      <PRICE>10.20</PRICE>
      <YEAR>1999</YEAR>
   </CD>
   <CD>
      <TITLE>For the good times</TITLE>
      <ARTIST>Kenny Rogers</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Mucik Master</COMPANY>
      <PRICE>8.70</PRICE>
      <YEAR>1995</YEAR>
   </CD>
   <CD>
      <TITLE>Big Willie style</TITLE>
      <ARTIST>Will Smith</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1997</YEAR>
   </CD>
   <CD>
      <TITLE>Tupelo Honey</TITLE>
      <ARTIST>Van Morrison</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1971</YEAR>
   </CD>
   <CD>
      <TITLE>Soulsville</TITLE>
      <ARTIST>Jorn Hoel</ARTIST>
      <COUNTRY>Norway</COUNTRY>
      <COMPANY>WEA</COMPANY>
      <PRICE>7.90</PRICE>
```

```
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Stop</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Bridge of Spies</TITLE>
    <ARTIST>T'Pau</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Siren</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Private Dancer</TITLE>
    <ARTIST>Tina Turner</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Capitol</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1983</YEAR>
  </CD>
  <CD>
    <TITLE>Midt om natten</TITLE>
    <ARTIST>Kim Larsen</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Medley</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1983</YEAR>
  </CD>
  <CD>
    <TITLE>Pavarotti Gala Concert</TITLE>
    <ARTIST>Luciano Pavarotti</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>DECCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1991</YEAR>
  </CD>
  <CD>
    <TITLE>The dock of the bay</TITLE>
    <ARTIST>Otis Redding</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Picture book</TITLE>
    <ARTIST>Simply Red</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Elektra</COMPANY>
    <PRICE>7.20</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Red</TITLE>
    <ARTIST>The Communards</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>London</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Unchain my heart</TITLE>
    <ARTIST>Joe Cocker</ARTIST>
    <COUNTRY>USA</COUNTRY>
```

```
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
   </CD>
</CATALOG>";

echo $ficheroxml;
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest) {
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()
```

```
////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

catalogo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<CATALOG>
   <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1988</YEAR>
   </CD>
   <CD>
      <TITLE>Greatest Hits</TITLE>
      <ARTIST>Dolly Parton</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>RCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1982</YEAR>
   </CD>
   <CD>
      <TITLE>Still got the blues</TITLE>
      <ARTIST>Gary Moore</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Virgin records</COMPANY>
      <PRICE>10.20</PRICE>
      <YEAR>1990</YEAR>
   </CD>
   <CD>
      <TITLE>Eros</TITLE>
      <ARTIST>Eros Ramazzotti</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>BMG</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1997</YEAR>
   </CD>
   <CD>
      <TITLE>One night only</TITLE>
      <ARTIST>Bee Gees</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1998</YEAR>
   </CD>
   <CD>
      <TITLE>Sylvias Mother</TITLE>
      <ARTIST>Dr.Hook</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS</COMPANY>
      <PRICE>8.10</PRICE>
      <YEAR>1973</YEAR>
   </CD>
   <CD>
      <TITLE>Maggie May</TITLE>
      <ARTIST>Rod Stewart</ARTIST>
      <COUNTRY>UK</COUNTRY>
```

```
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Romanza</TITLE>
    <ARTIST>Andrea Bocelli</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>When a man loves a woman</TITLE>
    <ARTIST>Percy Sledge</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>Savage Rose</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
  </CD>
  <CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
  </CD>
  <CD>
    <TITLE>Soulsville</TITLE>
    <ARTIST>Jorn Hoel</ARTIST>
    <COUNTRY>Norway</COUNTRY>
    <COMPANY>WEA</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
```

```
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
   </CD>
   <CD>
      <TITLE>Bridge of Spies</TITLE>
      <ARTIST>T'Pau</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Siren</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Private Dancer</TITLE>
      <ARTIST>Tina Turner</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Capitol</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1983</YEAR>
   </CD>
   <CD>
      <TITLE>Midt om natten</TITLE>
      <ARTIST>Kim Larsen</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Medley</COMPANY>
      <PRICE>7.80</PRICE>
      <YEAR>1983</YEAR>
   </CD>
   <CD>
      <TITLE>Pavarotti Gala Concert</TITLE>
      <ARTIST>Luciano Pavarotti</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>DECCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1991</YEAR>
   </CD>
   <CD>
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Picture book</TITLE>
      <ARTIST>Simply Red</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Elektra</COMPANY>
      <PRICE>7.20</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Red</TITLE>
      <ARTIST>The Communards</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>London</COMPANY>
      <PRICE>7.80</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Unchain my heart</TITLE>
      <ARTIST>Joe Cocker</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
   </CD>
</CATALOG>
```

En la función **`iniciar()`**, le hemos dicho que cargue de forma asíncrona, empleando el método **`GET`**, el fichero `datosXML.php`. Esta aplicación PHP, nos devolverá un fichero XML, con una lista de Cd de música con el artista, país, compañía, etc.

Instrucción de carga del fichero datosXML.php: `cargarAsync("datosXML.php");`

Si queremos cargar directamente un fichero XML, y conocemos su nombre, podremos escribir directamente:

```
cargarAsync("catalogo.XML");
```

En la función de `estadoPeticion()`, cuando `readyState` es 4 y el `status` es **OK** (200), accedemos a los resultados de la petición AJAX, en la propiedad `responseXML`. Para gestionar los datos XML, tendremos que recorrerlos empleando los métodos del DOM, ya que un fichero XML comparte la estructura en árbol de un documento HTML, y podemos utilizar, por tanto, los mismos métodos que empleamos para recorrer el DOM HTML.

En nuestro caso, lo primero que vamos a hacer es recorrer los elementos, que son los que contienen toda la información referente a los cd's de música:

```
// Almacenamos el fichero XML en la variable resultados.
resultados=this.responseXML;

// Tenemos que recorrer el fichero XML empleando los métodos del DOM
// Array que contiene todos los CD's del fichero XML
CDs= resultados.documentElement.getElementsByTagName("CD");
```

Haremos un bucle para recorrer todos los cd's del catálogo, y dentro de cada uno, imprimiremos los datos que nos interesen:

```
// Hacemos un bucle para recorrer todos los elementos CD.
for (i=0;i<CDs.length;i++){
…..
```

Dentro de cada CD, accederemos al elemento que nos interese e imprimiremos su contenido. Para imprimir el contenido de cada nodo, tendremos que hacerlo con el comando `try { } catch {}`, ya que si intentamos acceder a un nodo que no tenga contenido, nos dará un error de JavaScript, puesto que el elemento hijo no existe, y entonces se detendrá la ejecución de JavaScript y no imprimirá nuestro listado.

```
// Para cada CD leemos el título
titulos=CDs[i].getElementsByTagName("TITLE");

try{
   // Intentamos acceder al contenido de ese elemento
   salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
}catch (er){
   // En el caso de que no tenga contenido ese elemento imprimimos un espacio en blanco.
   salida+= "<td> </td>";
}
```

2.6.- Recepción de datos en formato JSON (parte I).

Otro formato de intercambio muy utilizado en AJAX, es el formato JSON. JSON es un formato de intercambio de datos, alternativo a XML, mucho más simple de leer, escribir e interpretar. Significa **JavaScript Object Notation**, y consiste en escribir los datos en formato de Javascript. Vamos a hacer un poco de repaso:

Arrays

Se pueden crear con corchetes:

```
var Beatles = ["Paul","John","George","Ringo"];
```

Con `new Array()`:

```
var Beatles = new Array("Paul","John","George","Ringo");
```

O también de la siguiente forma:

```
var Beatles = { "Paul","John","George","Ringo");
```

Objetos

Un objeto literal se puede crear entre llaves: `{ propiedad1:valor, propiedad2:valor, propiedad3:valor}`

```
var Beatles = {
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll"
}
```

Que será equivalente a:

```
var Beatles = new Object();

Beatles.Country = "England";
Beatles.YearFormed = 1959;
Beatles.Style = "Rock'n'Roll";
```

Y para acceder a sus propiedades lo podemos hacer con:

```
alert(Beatles.Style); // Notación de puntos
alert(Beatles["Style"]); // Notación de corchetes
```

Los objetos también pueden contener arrays literales: [...]

```
var Beatles = {
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll",
 "Members" : ["Paul","John","George","Ringo"]
}
```

Y los arrays literales podrán contener objetos literales a su vez: {…} , {...}

```
var Rockbands =
[
{  "Name" : "Beatles",  "Country" : "England",  "YearFormed" : 1959,  "Style" : "Rock'n'Roll",
"Members" :
 ["Paul","John","George","Ringo"] } , {  "Name" : "Rolling Stones",  "Country" : :England",
 "YearFormed" : 1962,  "Style" : "Rock'n'Roll", "Members" : ["Mick","Keith","Charlie","Bill"]
}
]
```

La sintaxis de JSON es como la sintaxis literal de un objeto, excepto que, esos objetos no pueden ser asignados a una variable. JSON representará los datos en sí mismos. Por lo tanto el objeto *Beatles* que vimos antes se definiría de la siguiente forma:

```
{
 "Name" : "Beatles",
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll",
 "Members" : ["Paul","John","George","Ringo"]
}
```

Una cadena JSON es, simplemente, una cadena de texto, y no un objeto en sí misma. Necesita ser convertida a un objeto antes de poder ser utilizada en JavaScript. Ésto se puede hacer con la función `eval()` de JavaScript y también se pueden usar lo que se conoce como analizadores JSON, que facilitarán esa conversión.

2.7.- Recepción de datos en formato JSON (parte II).

Veamos un ejemplo completo en el que recibimos, con AJAX, los datos procedentes de un listado de una tabla MySQL en formato JSON.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.7 - JSON Ajax asíncrono</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////


function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos los datos XML de forma asíncrona.
   // En este caso ponemos una página PHP que nos devuelve datos en formato XML
   // pero podríamos poner también el nombre de un fichero XML directamente: catalogos.xml
   // Se adjunta ejemplo de fichero XML.
   cargarAsync("datosjson.php");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada a cada cambio de estado de la petición AJAX
// cuando la respuesta del servidor es 200(fichero encontrado) y el estado de
// la solicitud es 4, accedemos a la propiedad responseText
////////////////////////////////////////////////////////
function estadoPeticion(){
   if (this.readyState==4 && this.status == 200){
      // Los datos JSON los recibiremos como texto en la propiedad this.responseText

      // Con la función eval() evaluaremos ese resultado para convertir a objetos y variables
el string que estamos recibiendo en JSON.
      // Lo que recibimos en formato JSON es un string que representa un array [ ... ] que
contiene objetos literales {...},{...},...
      /*            Ejemplo:           [            {"id":"2","nombrecentro":"IES            A
Piringalla","localidad":"Lugo","provincia":"Lugo","telefono":"982212010","fechavisita":"2010-
11-26","numvisitantes":"85"} , {"id":"10","nombrecentro":"IES As Fontiñas","localidad" : .....
} ]  */

      // Asignamos a la variable resultados la evaluación de responseText
```

```
      var resultados=eval( '(' +this.responseText+')');

      texto                  =                  "<table                  border=1><tr><th>Nombre
Centro</th><th>Localidad</th><th>Provincia</th><th>Telefono</th><th>Fecha
Visita</th><th>Numero Visitantes</th></tr>";
      // Hacemos un bucle para recorrer todos los objetos literales recibidos en el array
resultados y mostrar su contenido.
      for (var i=0; i < resultados.length; i++){
         objeto = resultados[i];
         texto+="<tr><td>"+objeto.nombrecentro+"</td><td>"+
            objeto.localidad+"</td><td>"+objeto.provincia+"</td><td>"+
            objeto.telefono+"</td><td>"+objeto.fechavisita+"</td><td>"+
            objeto.numvisitantes+"</td></tr>";
      }

      // Desactivamos el indicador AJAX cuando termina la petición
      document.getElementById("indicador").innerHTML="";

      // Imprimimos la tabla dentro del contenedor resultados.
      document.getElementById("resultados").innerHTML=texto;
   }
}
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
```

```
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

datosjson.php

```
<?php
   // Cabecera para indicar que vamos a enviar datos JSON y que no haga caché de los datos.
   header('Content-Type: application/json');
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   /*
      Utilizar el fichero dbcreacion.sql incluído en la carpeta para crear la base de datos,
      usuario y tabla en tu servidor MySQL.
      Si fuera necesario modifica los datos de la configuración y adáptalos a tu entorno
      de trabajo.
   */

   // Configuración BASE DE DATOS MYSQL
   $servidor = "localhost";
   $basedatos = "ajax";
   $usuario = "ajax";
   $password = "dwec";

   // Creamos la conexión al servidor.
   $conexion=mysql_connect($servidor, $usuario, $password) or die(mysql_error());
   mysql_query("SET NAMES 'utf8'",$conexion);

   // Seleccionar la base de datos en esa conexion.
   mysql_select_db($basedatos,$conexion) or die(mysql_error());

   // Consulta SQL para obtener los datos de los centros.
   $sql="select * from centros order by nombrecentro";
   $resultados=mysql_query($sql,$conexion) or die(mysql_error());

   while ( $fila = mysql_fetch_array($resultados, MYSQL_ASSOC))
   {
      // Almacenamos en un array  cada una de las filas que vamos leyendo del recordset.
      $datos[]=$fila;
   }

   // Como resultado se puede enviar algo similar a:
   /*
      [      {"id":"3","nombrecentro":"IES      San      Clemente","localidad":"Santiago      de
Compostela","provincia":"A      Coruña"      ,"telefono":"981580496","fechavisita":"2010-11-26",
"numvisitantes":"60"} , {"id":"10","nombrecentro":"IES As Fontiñas","localidad" : ..... } ]

      // Empleando la siguiente instrucción:
      echo json_encode($datos);
   */

   /*
      O si queremos enviar como resultado un array de objetos literales llamado
```

```
      resultados, haremos:
      resultados               =               [{"id":"2","nombrecentro":"IES               A
Piringalla","localidad":"Lugo","provincia":"Lugo","telefono":"982212010","fechavisita":"2010-
11-26","numvisitantes":"85"} , {"id":"10","nombrecentro":"IES As Fontiñas","localidad" : .....
}]

      Empleando la siguiente instrucción:
      echo "resultados=".json_encode($datos);

      De esta forma en la página index.js sólo tendríamos que poner
      eval('(' + this.responseText + ')')
      y así ya tenemos disponible en JavaScript una variable resultados que es un array
      que contendrá los objetos literales.
   */

   echo json_encode($datos); // función de PHP que convierte a formato JSON el array.

   mysql_close($conexion);
?>
```

dbcreacion.sql

```
CREATE USER 'ajax'@'localhost' IDENTIFIED BY 'dwec';

GRANT USAGE ON * . * TO 'ajax'@'localhost' IDENTIFIED BY 'dwec' WITH MAX_QUERIES_PER_HOUR 0
MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;

CREATE DATABASE IF NOT EXISTS `ajax` ;

GRANT ALL PRIVILEGES ON `ajax` . * TO 'ajax'@'localhost';

use  `ajax`;


CREATE TABLE IF NOT EXISTS `centros` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `nombrecentro` varchar(150) NOT NULL,
  `localidad` varchar(100) NOT NULL,
  `provincia` varchar(50) NOT NULL,
  `telefono` varchar(9) NOT NULL,
  `fechavisita` date NOT NULL,
  `numvisitantes` int(10) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=11 ;

--
-- Volcar la base de datos para la tabla `centros`
--

INSERT  INTO  `centros`  (`id`,  `nombrecentro`,  `localidad`,  `provincia`,  `telefono`,
`fechavisita`, `numvisitantes`) VALUES
(1, 'IES Ramon Mª Aller Ulloa', 'Lalin', 'Pontevedra', '986780114', '2010-11-26', 90),
(2, 'IES A Piringalla', 'Lugo', 'Lugo', '982212010', '2010-11-26', 85),
(3, 'IES San Clemente', 'Santiago de Compostela', 'A Coruña', '981580496', '2010-11-26', 60),
(4, 'IES de Teis', 'Vigo', 'Pontevedra', '986373811', '2010-11-27', 72),
(5, 'IES Leliadoura', 'Ribeira', 'A Coruña', '981874633', '2010-11-25', 0),
(6, 'IES Cruceiro Baleares', 'Culleredo', 'A Coruña', '981660700', '2010-11-26', 30),
(7, 'IES Leliadoura', 'Ribeira', 'A Coruña', '981874633', '2010-11-25', 50),
(8, 'IES Cruceiro Baleares', 'Culleredo', 'A Coruña', '981660700', '2010-11-26', 30),
(9, 'IES As Lagoas', 'Ourense', 'Ourense', '988391325', '2010-11-26', 35),
(10, 'IES As Fontiñas', 'Santiago de Compostela', 'A Coruña', '981573440', '2010-11-27', 64);
```

catalogo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<CATALOG>
   <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
```

```
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
</CD>
<CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
</CD>
<CD>
    <TITLE>Still got the blues</TITLE>
    <ARTIST>Gary Moore</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Virgin records</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1990</YEAR>
</CD>
<CD>
    <TITLE>Eros</TITLE>
    <ARTIST>Eros Ramazzotti</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>BMG</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
</CD>
<CD>
    <TITLE>One night only</TITLE>
    <ARTIST>Bee Gees</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1998</YEAR>
</CD>
<CD>
    <TITLE>Sylvias Mother</TITLE>
    <ARTIST>Dr.Hook</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS</COMPANY>
    <PRICE>8.10</PRICE>
    <YEAR>1973</YEAR>
</CD>
<CD>
    <TITLE>Maggie May</TITLE>
    <ARTIST>Rod Stewart</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
</CD>
<CD>
    <TITLE>Romanza</TITLE>
    <ARTIST>Andrea Bocelli</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
</CD>
<CD>
    <TITLE>When a man loves a woman</TITLE>
    <ARTIST>Percy Sledge</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1987</YEAR>
</CD>
<CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>Savage Rose</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
</CD>
<CD>
```

```
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
  </CD>
  <CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
  </CD>
  <CD>
    <TITLE>Soulsville</TITLE>
    <ARTIST>Jorn Hoel</ARTIST>
    <COUNTRY>Norway</COUNTRY>
    <COMPANY>WEA</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Stop</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Bridge of Spies</TITLE>
    <ARTIST>T'Pau</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Siren</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Private Dancer</TITLE>
    <ARTIST>Tina Turner</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Capitol</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1983</YEAR>
  </CD>
  <CD>
    <TITLE>Midt om natten</TITLE>
    <ARTIST>Kim Larsen</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Medley</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1983</YEAR>
```

```
   </CD>
   <CD>
      <TITLE>Pavarotti Gala Concert</TITLE>
      <ARTIST>Luciano Pavarotti</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>DECCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1991</YEAR>
   </CD>
   <CD>
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Picture book</TITLE>
      <ARTIST>Simply Red</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Elektra</COMPANY>
      <PRICE>7.20</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Red</TITLE>
      <ARTIST>The Communards</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>London</COMPANY>
      <PRICE>7.80</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Unchain my heart</TITLE>
      <ARTIST>Joe Cocker</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
   </CD>
</CATALOG>
```

Si quieres probar el ejemplo, necesitas un servidor web, PHP y MySQL. Puedes instalarte por ejemplo XAMPP en cualquiera de sus versiones para Windows o Linux.

En este ejemplo, se realiza una consulta a una tabla (se adjunta código SQL de creación de la base de datos, usuario, contraseña y la tabla con los datos). La página PHP, devolverá los resultados en una cadena de texto, que contiene un array de objetos literales en formato JSON.

Código de PHP del fichero `datosJSON.php`:

```
….
// Consulta SQL para obtener los datos de los centros.
$sql="select * from centros order by nombrecentro";
$resultados=mysql_query($sql,$conexion) or die(mysql_error());

while ( $fila = mysql_fetch_array($resultados, MYSQL_ASSOC)){
   // Almacenamos en un array  cada una de las filas que vamos leyendo del recordset.
   $datos[]=$fila;
}

echo JSON_encode($datos); // función de PHP que convierte a formato JSON el array.
….
```

Nuestra aplicación de JavaScript recibe, por AJAX, esos datos, en la propiedad **`responseText`**. Para poder utilizar directamente esos datos en JavaScript, lo que tenemos que hacer es evaluar la expresión (cadena de texto, que recibimos de la página `datosJSON.php`). La expresión JSON contenía

un array de objetos literales, por lo que tenemos que crear una variable, para asignar ese array y poder recorrerlo.

Para evaluar la expresión lo hacemos con la función `eval()` de JavaScript:

```
// Asignamos a la variable resultados la evaluación de responseText
var resultados=eval( '(' +this.responseText+')');

// Hacemos un bucle para recorrer todos los objetos literales recibidos en el array resultados
y mostrar su contenido.
for (var i=0; i < resultados.length; i++){
     objeto = resultados[i];
     texto+="<tr><td>"+objeto.nombrecentro+
         "</td><td>"+objeto.localidad+"</td><td>"+
         objeto.provincia+"</td><td>"+
         objeto.telefono+"</td><td>"+
         objeto.fechavisita +"</td><td>"+
         objeto.numvisitantes+ "</td></tr>";
}
```

"Si vivir es durar, prefiero una canción de los Beatles a un Long Play de los Boston Pops. - Mafalda."
Quino, Joaquín Salvador Lavado

3.- Librerías cross-browser para programación AJAX.

Caso práctico

*El estudio del objeto para trabajar con AJAX, está comenzando a dar sus frutos. Lo que más le fastidia a **Antonio**, es que necesita programar bastante código, y aunque puede crear alguna librería para acelerar la programación, también ve que las diferentes incompatibilidades, entre navegadores, no van a ayudar nada en esta labor. Por esta razón está un poco desilusionado, por que le va a suponer bastante trabajo, aunque los resultados merecen la pena.*

*En ese momento llega **Juan**, y le da la sorpresa que le había comentado hace unos días. Le facilita un pequeño tutorial sobre la librería jQuery, que le va a permitir hacer peticiones AJAX utilizando prácticamente una línea de código. No tendrá que preocuparse por temas de cross-browsing y, además, la misma librería le facilitará métodos para hacer todo tipo de efectos, animaciones, etc. Esta librería cuenta además con infinidad de complementos gratuitos, que permiten hacer prácticamente cualquier cosa en muy poco tiempo.*

La programación con AJAX, es uno de los pilares de lo que se conoce como web 2.0, término que incluye a las aplicaciones web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración web. Ejemplos de la web 2.0, pueden ser las comunidades web, los servicios web, aplicaciones web, redes sociales, servicios de alojamiento de vídeos, wikis, blogs, mashup (*página web o aplicación que usa y combina datos, presentaciones y funcionalidad procedentes de una o más fuentes para crear nuevos servicios. El término implica integración fácil y rápida, usando a menudo APIs abiertos y fuentes de datos con el objetivo de producir resultados enriquecidos combinando diferentes fuentes*), etc.

Más información sobre Mashup (aplicaciones web híbridas).

http://es.wikipedia.org/wiki/Mashup_%28aplicaci%C3%B3n_web_h%C3%ADbrida%29

Gracias a las aplicaciones web 2.0, se han desarrollado gran cantidad de utilidades/herramientas/frameworks para el desarrollo web con JavaScript, DHTML(HTML dinámico) y AJAX. La gran ventaja de usar alguna librería o framework para AJAX, es la del ahorro de tiempo y código, en nuestras aplicaciones. Veremos que con algunas librerías vamos a realizar peticiones AJAX, con una simple instrucción de código sin tener que preocuparnos de crear el objeto XHR, ni gestionar el código de respuesta del servidor, los estados de la solicitud, etc.

Otra de las ventajas que nos aportan este tipo de librerías, es la de la compatibilidad entre navegadores (cross-browser). De esta forma tenemos un problema menos, ya que la propia librería será capaz de crear la petición AJAX de una forma u otra, dependiendo del navegador que estemos utilizando.

A principios del año 2008 Google liberó su API de librerías AJAX, como una red de distribución de contenido y arquitectura de carga, para algunos de los frameworks más populares. Mediante esta API se eliminan las dificultades a la hora de desarrollar mashups en JavaScript. Se elimina el problema de alojar las librerías (ya que están centralizadas en Google), configurar las cabeceras de cache, etc. Esta API ofrece acceso a las siguientes librerías Open Source, realizadas con JavaScript:

- ✓ jQuery.
- ✓ prototype.
- ✓ scriptaculous.
- ✓ mooTools.
- ✓ dojo, swfobject, chrome-frame, webfont, etc.

Los scripts de estas librerías están accesibles directamente utilizando la URL de descarga, o a través del método *google.load*() del cargador de la API AJAX de Google.

Hay muchísimas librerías que se pueden utilizar para programar AJAX, dependiendo del lenguaje que utilicemos. Mira el siguiente anexo para comprobarlo

Nosotros nos vamos a centrar en el uso de la librería jQuery, por ser una de las más utilizadas hoy en día por empresas como Google, DELL, digg, NBC, CBS, NETFLIX, mozilla.org, wordpress, drupal, etc.

3.1.- Introducción a jQuery (parte I).

jQuery es un framework JavaScript, que nos va a simplificar muchísimo la programación. Como bien sabes, cuando usamos JavaScript tenemos que preocuparnos de hacer scripts compatibles con varios navegadores y, para conseguirlo, tenemos que programar código compatible.

jQuery nos puede ayudar muchísimo a solucionar todos esos problemas, ya que nos ofrece la infraestructura necesaria para crear aplicaciones complejas en el lado del cliente. Basado en la filosofía de "*escribe menos y produce más*", entre las ayudas facilitadas por este framework están: la creación de interfaces de usuario, uso de efectos dinámicos, AJAX, acceso al DOM, eventos, etc. Además esta librería cuenta con infinidad de plugins, que nos permitirán hacer presentaciones con imágenes, validaciones de formularios, menús dinámicos, drag-and-drop, etc.

Esta librería es gratuita, y dispone de licencia para ser utilizada en cualquier tipo de plataforma, personal o comercial. El fichero tiene un tamaño aproximado de 31 KB, y su carga es realmente rápida. Además, una vez cargada la librería, quedará almacenada en caché del navegador, con lo que el resto de páginas que hagan uso de la librería, no necesitarán cargarla de nuevo desde el servidor.

Página Oficial de descarga de la librería jQuery.
http://jquery.com/

Documentación oficial de la librería jQuery.
http://docs.jquery.com/Main_Page

Para poder programar con jQuery, lo primero que tenemos que hacer es cargar la librería. Para ello, podemos hacerlo de dos formas:

Cargando la librería directamente desde la propia web de jQuery con la siguiente instrucción:

```
<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
```

De esta forma, siempre nos estaremos descargando la versión más actualizada de la librería. El único inconveniente, es que necesitamos estar conectados a Internet para que la librería pueda descargarse.

Cargando la librería desde nuestro propio servidor:

```
<script type="text/javascript" src="jquery.js"></script>
```

De esta forma, el fichero de la librería estará almacenado como un fichero más de nuestra aplicación, por lo que no necesitaremos tener conexión a Internet (si trabajamos localmente), para poder usar la librería. Para poder usar este método, necesitaremos descargarnos el fichero de la librería desde la página de jQuery (`jquery.com`). Disponemos de dos versiones de descarga: la *versión de producción* (comprimida para ocupar menos tamaño), y *la versión de desarrollo* (descomprimida). Generalmente descargaremos la versión de producción, ya que es la que menos tamaño ocupa. La versión de desarrollo tiene como única ventaja que nos permitirá leer, con más claridad, el código fuente de la librería (si es que estamos interesados en modificar algo de la misma).

La clave principal para el uso de jQuery radica en el uso de la función `$()`, que es un alias de `jQuery()`. Esta función se podría comparar con el clásico `document.getElementById()`, pero con una

diferencia muy importante, ya que soporta selectores CSS, y puede devolver arrays. Por lo tanto `$()` es una versión mejorada de `document.getElementById()`.

Selectores CSS.

Anexo II - Selectores CSS que deberías conocer

Esta función `$("selector")`, acepta como parámetro una cadena de texto, que será un selector CSS, pero también puede aceptar un segundo parámetro, que será el contexto en el cuál se va a hacer la búsqueda del selector citado. Otro uso de la función, puede ser el de `$(function){..});` equivalente a la instrucción `$(document).ready (function() {...});` que nos permitirá detectar cuando el DOM está completamente cargado.

Verás un ejemplo de cómo usar estas instrucciones en apartado siguiente 3.2.

3.2.- Introducción a jQuery (parte II).

Vamos a ver en este apartado, un ejemplo programado por el método tradicional, y su equivalencia, usando la librería jQuery:

normal.html

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="utf-8">
      <title>Ejemplo sin jQuery</title>
      <style type="text/css">
         .colorido{
            background-color:#99FF33;
         }
      </style>

      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript">
         ////////////////////////////////////////////////////////////////////
         // Cuando el documento esté cargado completamente llamamos a la función iniciar().
         ////////////////////////////////////////////////////////////////////
         crearEvento(window,"load",iniciar);
         ////////////////////////////////////////////////////////

         function iniciar(){
            var tabla=document.getElementById("mitabla"); // Seleccionamos la tabla.
            var filas= tabla.getElementsByTagName("tr");   // Seleccionamos las filas de la
tabla.

            for (var i=0; i<filas.length; i++){
               if (i%2==1){
                  // Es una fila impar
                  // Aplicamos la clase .colorido a esas filas.
                  filas[i].setAttribute('class','colorido');
               }
            }
         }
      </script>
   </head>
   <body>
      <table width="200" border="1" align="center" id="mitabla">
         <tr>
            <td><div align="center"><strong>pais</strong></div></td>
            <td><div align="center"><strong>habitantes</strong></div></td>
            <td><div align="center"><strong>renta</strong></div></td>
         </tr>
         <tr>
            <td>España</td>
            <td>15600000</td>
            <td>25000</td>
         </tr>
         <tr>
            <td>Italia</td>
            <td>45105500</td>
            <td>45000</td>
```

```
         </tr>
         <tr>
            <td>Francia</td>
            <td>58454545</td>
            <td>45645</td>
         </tr>
         <tr>
            <td>Uk</td>
            <td>78799788</td>
            <td>88547</td>
         </tr>
         <tr>
            <td>USA</td>
            <td>98878787</td>
            <td>45124</td>
         </tr>
      </table>
   </body>
</html>
```

jquery.html

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="utf-8">
      <title>Ejemplo con jQuery</title>
      <style type="text/css">
         .colorido{
            background-color:#99FF33;
         }
      </style>

      <script type="text/javascript" src="http://code.jquery.com/jquery-latest.js"></script>
      <script type="text/javascript">
         $(document).ready(function(){
            // Cuando el documento esté preparado se ejecuta esta función.
            // Seleccionamos las filas impares contenidas dentro de mitabla y le aplicamos la
clase colorido.
            $("#mitabla tr:nth-child(even)").addClass("colorido");
         });
      </script>
   </head>
   <body>
      <table width="200" border="1" align="center" id="mitabla">
         <tr>
            <td><div align="center"><strong>pais</strong></div></td>
            <td><div align="center"><strong>habitantes</strong></div></td>
            <td><div align="center"><strong>renta</strong></div></td>
         </tr>
         <tr>
            <td>España</td>
            <td>15600000</td>
            <td>25000</td>
         </tr>
         <tr>
            <td>Italia</td>
            <td>45105500</td>
            <td>45000</td>
         </tr>
         <tr>
            <td>Francia</td>
            <td>58454545</td>
            <td>45645</td>
         </tr>
         <tr>
            <td>Uk</td>
            <td>78799788</td>
            <td>88547</td>
         </tr>
         <tr>
            <td>USA</td>
            <td>98878787</td>
            <td>45124</td>
         </tr>
      </table>
   </body>
</html>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
```

```
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

Ejemplo usando el método tradicional con JavaScript:

```
     … Aquí irán las cabeceras y clase .colorido

<script type="text/javascript" src="funciones.js"></script>
<script type="text/javascript">
   ////////////////////////////////////////////////////////////////////
   // Cuando el documento esté cargado completamente llamamos a la función iniciar().
   ////////////////////////////////////////////////////////////////////
   crearEvento(window,"load",iniciar);
   ////////////////////////////////////////////////////////////////////

   function iniciar(){
      var tabla=document.getElementById("mitabla"); // Seleccionamos la tabla.
      var filas= tabla.getElementsByTagName("tr");  // Seleccionamos las filas de la tabla.

      for (var i=0; i<filas.length; i++){
         if (i%2==1){
            // Es una fila impar
            // Aplicamos la clase .colorido a esas filas.
            filas[i].setAttribute('class','colorido');
         }
      }
   }
</script>
</head>
<body>
     .. Aquí irá la tabla …
</body>
</HTML>
```

Ejemplo equivalente al anterior, programado usando la librería jQuery:

```
     … Aquí irán las cabeceras y clase .colorido

<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
// También podríamos poner $(function) {….});
$(document).ready(function()        // Cuando el documento esté preparado se ejecuta esta
función.
{
     // Seleccionamos las filas impares contenidas dentro de mitabla y le aplicamos la clase
colorido.
     $("#mitabla tr:nth-child(even)").addClass("colorido");
});
</script>
</head>
<body>
     .. Aquí irá la tabla …
</body>
</HTML>
```

Como puedes observar, la reducción de código es considerable. Con 2 instrucciones, hemos conseguido lo mismo, que hicimos en el ejemplo anterior con 7 (sin contar el código de `crearEvento` del fichero `funciones.js`).

3.3.- Función $.ajax() en jQuery.

La principal función para realizar peticiones AJAX en jQuery es **`$.AJAX()`** (importante no olvidar el punto entre `$` y `AJAX()`). Ésta es una función de bajo nivel, lo que quiere decir que disponemos de la posibilidad de configurar, prácticamente todos los parámetros de la petición AJAX, y será, por tanto, equivalente a los métodos clásicos que usamos en la programación tradicional.

La sintaxis de uso es: **`$.AJAX(opciones)`**
En principio, esta instrucción parece muy simple, pero el número de opciones disponibles, es relativamente extenso. Ésta es la estructura básica:

```
$.AJAX({
     url: [URL],
```

```
     type: [GET/POST],
     success: [function callback exito(data)],
     error: [function callback error],
     complete: [function callback error],
     ifModified: [bool comprobar E-Tag],
     data: [mapa datos GET/POST],
     async: [bool que indica sincronía/asincronia]
});
```

Por ejemplo:

```
$.AJAX({
     url: '/ruta/pagina.php',
     type: 'POST',
     async: true,
     data: 'parametro1=valor1&parametro2=valor2',
     success: function (respuesta)
     {
          alert(respuesta);
     },
     error: mostrarError
});
```

Veamos algunas propiedades de la función `$.AJAX()` de jQuery:

Nombre	Tipo	Descripción
`url`	String	La URL a la que se le hace la petición AJAX.
`type`	String	El método HTTP a utilizar para enviar los datos: POST o GET. Si se omite se usa GET por defecto.
`data`	Object	Un objeto en el que se especifican parámetros que se enviarán en la solicitud. Si es de tipo GET, los parámetros irán en la URL. Si es POST, los datos se enviarán en las cabeceras. Es muy útil usar la función serialize(), para construir la cadena de datos.
`dataType`	String	Indica el tipo de datos que se espera que se devuelvan en la respuesta: XML, HTML, JSON, JSONp, script, text (valor por defecto en el caso de omitir dataType).
`success`	Function	Función que será llamada, si la respuesta a la solicitud terminó con éxito.
`error`	Function	Función que será llamada, si la respuesta a la solicitud devolvió algún tipo de error.
`complete`	Function	Función que será llamada, cuando la solicitud fue completada.

Todos los parámetros de uso de la función $.AJAX() de jQuery.
http://api.jquery.com/jQuery.AJAX/

3.4.- El método .load() y las funciones $.post() , $.get() y $.getJSON() en jQuery.

La función `$.AJAX()` es una función muy completa, y resulta bastante pesada de usar. Su uso es recomendable, para casos muy concretos, en los que tengamos que llevar un control exhaustivo de la petición AJAX. Para facilitarnos el trabajo, se crearon 3 funciones adicionales de alto nivel, que permiten realizar peticiones y gestionar las respuestas obtenidas del servidor:

El método .load()

Este método, es la forma más sencilla de obtener datos desde el servidor, ya que de forma predeterminada, los datos obtenidos son cargados en el objeto al cuál le estamos aplicando el método.

Su sintaxis es: `.load( url, [datos], [callback] )`

La función `callback` es opcional, y es ahí donde pondremos la función de retorno, que será llamada una vez terminada la petición. En esa función realizaremos tareas adicionales, ya que la acción por defecto de cargar en un objeto el contenido devuelto en la petición, la realiza el propio método `load()`.

Ejemplos:

```
$("#noticias").load("feeds.HTML");
// carga en el contenedor con id noticias lo que devuelve la página feeds.HTML.

$("#objectID").load("test.php", { 'personas[]': ["Juan", "Susana"] } );
// Pasa un array de datos al servidor con el nombre de dos personas.
```

Cuando se envían datos en este método, se usará el método `POST`. Si no se envían datos en la petición, se usará el método `GET`.

Más información sobre el método `.load()` en jQuery.
http://api.jquery.com/jQuery.post/

La función $.post()

Nos permite realizar peticiones AJAX al servidor, empleando el método `POST`. Su sintaxis es la siguiente:

```
$.post( url, [datos], [callback], [tipo] )
```

Ejemplos:

```
$.post("test.php");

$.post("test.php", { nombre: "Juana", hora: "11am" } );

$.post("test.php", function(resultados) {
     alert("Datos Cargados: " + resultados);
});
```

Más información sobre el método .post() en jQuery.
http://api.jquery.com/jQuery.post/

La función $.get() y $.getJSON()

Hacen prácticamente lo mismo que `POST`, y tienen los mismos parámetros, pero usan el método `GET` para enviar los datos al servidor. Si recibimos los datos en formato JSON, podemos emplear `$.getJSON()` en su lugar.

```
$.get( url, [datos], [callback], [tipo] ) | $.getJSON( url, [datos], [callback], [tipo] )
```

Más información sobre el método .get() en jQuery.
http://api.jquery.com/jQuery.get/

3.5.- Herramientas adicionales en programación AJAX.

Cuando programamos en AJAX, uno de los inconvenientes que nos solemos encontrar, es el de la detección de errores. Estos errores pueden venir provocados por fallos de programación en JavaScript, fallos en la aplicación que se ejecuta en el servidor, etc.

Para poder detectar estos errores, necesitamos herramientas que nos ayuden a encontrarlos. En la programación con JavaScript, los errores los podemos detectar con el propio navegador. Por ejemplo, en el navegador Firefox para abrir la consola de errores, lo podemos hacer desde el menú Herramientas, o bien pulsando las teclas **CTRL + Mayúsc. + J** (en Windows). En la consola, se nos mostrarán todos los errores que se ha encontrado durante la ejecución de la aplicación. En Internet Explorer versión 9, podemos abrir la **Herramienta de Desarrollo**, pulsando la tecla F12. Desde esta herramienta se pueden consultar los errores de JavaScript, activar los diferentes modos de compatibilidad entre versiones de este navegador, deshabilitar CSS, JavaScript, etc.

Para la detección de errores en AJAX, necesitamos herramientas adicionales o complementos. Para Firefox disponemos de un complemento denominadoFirebug. Este complemento nos va a permitir hacer infinidad de cosas: detectar errores de JavaScript, depurar código, analizar todo el DOM del documento en detalle, ver y modificar el código CSS, analizar la velocidad de carga de las páginas, etc. Además, también incorpora en su consola, la posibilidad de ver las peticiones AJAX que se están realizando al servidor: se pueden ver los datos enviados, su formato, los datos recibidos, los errores, etc. Si por ejemplo se produce algún tipo de error en la petición al servidor, en la consola podremos verlo y así poder solucionar ese fallo.

Vídeo sobre el uso de firebugs de firefox para ajax
Vídeo de introducción a node.js
Anexo III - 10 extensiones de Firefox para desarrollo web.

3.6.- Plugins jQuery.

La librería jQuery, incorpora funciones que nos van a ayudar muchísimo, en la programación de nuestras aplicaciones. Además de todo lo que nos aporta la librería, disponemos de plugins o añadidos que aportan funcionalidades avanzadas.

Vamos a encontrar plugins en un montón de categorías: AJAX, animación y efectos, DOM, eventos, formularios, integración, media, navegación, tablas, utilidades, etc.

Anexo IV - Efectos con jQuery.
Documentación oficial de jQuery. **http://docs.jquery.com/Main_Page**

Antes de poder usar cualquier plugin de jQuery, será necesario cargar primero la librería de jQuery, y a continuación la librería del plugin que deseemos, por ejemplo:

```
<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>

<script type="text/javascript" src="ejemploplugin.js"></script>
```

Todos los plugins contienen documentación, en la que se explica cómo usar el plugin.

Desde la web oficial de `jquery.com`, puedes hojear todos los plugins disponibles para jQuery:
Plugins para jQuery. **http://plugins.jquery.com/**

También es muy común el encontrar páginas, en las que se muestran rankings de los mejores plugins para jQuery. Algunos ejemplos pueden ser:
Los Mejores plugins jQuery del año 2011.
http://www.ajaxline.com/best-jquery-plugins-february-2011
Los 130 mejores plugins de jQuery.
http://pixelcurse.com/jquery/ultimate-roundup-of-best-jquery-plugins
Búsqueda en Google de los mejores plugins de jQuery.
http://www.google.es/#sclient=psy&hl=es&source=hp&q=best+jquery+plugins&aq=f&aqi=g3&aql=&oq=&pbx=1&bav=on.2,or.r_gc.r_pw.&fp=d64bad206e66ecae&biw=1920&bih=888

3.7.- Ejemplos en vídeo, de AJAX con jQuery.

Y para terminar, te vamos a poner unos enlaces a unos vídeos hospedados en Youtube.com

Gestión de una tabla con jQuery
http://www.youtube.com/watch?feature=player_embedded&v=oj-ktOyHzao

Trabajando con hiperenlaces
http://www.youtube.com/watch?feature=player_embedded&v=f5OEAlWx46M
Iconos indicando tipos de hiperenlace
http://www.youtube.com/watch?feature=player_embedded&v=dp4cIWHVWEc
Menú con efectos dinámicos
http://www.youtube.com/watch?feature=player_embedded&v=j5I5Dum4DA8
Efectos Rollover sobre imágenes
http://www.youtube.com/watch?feature=player_embedded&v=hC94Ks8SnFE

Y aquí tienes el código fuente:

fichero zip con los fuentes de todos los ejercicios de los vídeos

Anexo I - Listado de librerías, frameworks y herramientas para AJAX, DHTML y JavaScript

Con esto de las aplicaciones web 2.0, se han desarrollado una gran cantidad de utilidades/herramientas/framework para el desarrollo web con JavaScript, DHTML (HTML dínamico) y AJAX. He aquí el gran listado:

- ✓ Prototype es un *framework* basado en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web. Es una herramienta que implementa las técnicas AJAX y su potencial es aprovechado al máximo cuando se desarrolla con Ruby On Rails. (fuente)
- ✓ AHAH (Asychronous HTML and HTTP) es un microformato que permite la actualización asíncrona del contenido (X)HTML, y su formateo con CSS, al estilo de lo que hace AJAX. La diferencia con éste es que esto se realiza utilizando (X)HTML y no XML. Pero como (X)HTML puede ser visto como un dialecto de XML, entonces podemos decir que AHAH está incluído en AJAX (por lo que lo de llamarlo AJAX 2.0 es muy sensacionalista y poco estricto). (fuente)
- ✓ dojo es un Framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop APIs, widget APIs, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX. Dojo resuelve asuntos de usabilidad comunes como pueden ser la navegación y detección del navegador, soportar cambios de URL en la barra de URLs para luego regresar a ellas(bookmarking), y la habilidad de degradar cuando AJAX/JavaScript no es completamente soportado en el cliente. (fuente)
- ✓ AjaxAC es un marco de trabajo escrito en PHP y que utiliza AJAX para la relación con el servidor. Este *framework* es liberado bajo la licencia de Apache v2.0. (fuente)
- ✓ JSAN - JavaScript Archive Network es una colección de recursos para JavaScript de código abierto.
- ✓ Ajax.NET Professional es uno de las primeras librerías AJAX disponibles para Microsoft ASP.NET y trabaja con .NET 1.1 y 2.0. Puedes encontrar una guía rápida de cómo dar tus primeros pasos en Ajax.NET, en su web oficial.
- ✓ AjaxRequest Library es producto de AjaxToolbox.com, que simplifica y extiende las capacidades del objeto XMLHttpRequest (el corazón de AJAX) y te permite desarrollar tus proyectos, sin tener que preocuparte por los procesos a bajo nivel.
- ✓ ATLAS es un paquete de nuevas tecnologías de desarrollo web que integra un extenso conjunto de librerias "client script" con la rica plataforma de desarrollo del lado del servidor ASP .NET lo que nos va a permitir poder crear aplicaciones que tengan la posibilidad de realizar actualizaciones sobre una página web en el cliente haciendo llamadas directas al servidor Web sin la necesidad de hacer un "Refresco de Página", lo que nos permite poder aprovechar todo el potencial del lado del Servidor haciendo mucho trabajo en el Cliente permitiendo una mejor interacción de nuestros usuarios con los sistemas que desarrollemos. (fuente)
- ✓ Bajax es una pequeña y simple librería JavaScript para usar AJAX en nuestra páginas web. Es independiente del lenguaje de programación. Podemos mostrar contenido dinámico usando comandos simples. (mas info)
- ✓ MochiKit es una biblioteca de clases de propósito general escrita en JavaScript que suministra características de otros lenguajes de programación como Python u Objective-C. (fuente)
- ✓ Code Snippets es un repositorio público de códigos fuente. Permite facilmente crear tu colección personal de códigos/script, categorizarlas con tags y compartirlas con todo el mundo.
- ✓ DHTML API, Drag & Drop for Images and Layers librería JavaScript DHTML la cual agrega funciones de Drag Drop (arrastre/mover) sobre capas (layers) y cualquier imágen. Una librería que no debe faltarnos.
- ✓ DHTMLgoodies.com nos ofrece una gran cantidad de utilidades/scripts de DHTML, JavaScript y Ajax.
- ✓ Dynamic Drive un lugar en la web donde podemos obtener de manera gratuita utilidades/scripts DTHML y JavaScript para agregarlas a nuestros proyectos. Este sitio se actualiza regularmente.

- ✓ DynAPI es una librería, de código abierto, en JavaScript para crear componentes Dinámicos para HTML (DHTML) en una página web.
- ✓ qooxdoo es una librería que ofrece muchas facilidades para crear interfaces javascript avanzados, incluyendo una consola de depuración, manejo de eventos, control del foco… Soporta la mayoría de los navegadores actuales y tiene licencia LGPL. (fuente)
- ✓ Engine for Web Applications es un framework para desarrollo de aplicaciones web del lado del cliente.
- ✓ JavaScript Libraries sitio web donde podemos encontrar gran cantidad de utilidades/scripts en JavaScript y DHTML, tales como: manejo de formularios, retención de variables, cargar/mostrar imágenes, menús, efectos y entre otros como XML/RSS/DOM.
- ✓ Javascript Toolbox es un repositorio de códigos y librerías reutilizables que satisfacer necesidades comunes que enfrentan muchos desarrolladores web. La gran cantidad de estos código es compatible con la mayoría de navegadores. Podemos encontrar códigos fiables, pues son probados y testeados para un correcto funcionamiento. Excelente iniciativa realmente!.
- ✓ Taconite es framework que simplifica la creación de aplicaciones web Ajax. Automatiza las tediosas tareas relacionadas con Ajax, tales como la creación y gestión del objeto XMLHttpRequest y la creación de contenido dinámico. Taconite se puede utilizar con todos los navegadores web actuales (Firefox, Safari, Internet Explorer, Opera y Konqueror, por citar algunos) y puede utilizarse con tecnologías del lado del servidor como Java EE, .Net, PHP ó cualquier lenguaje que retorne como respuesta XHTML.
- ✓ jQuery es un nuevo tipo de librerias de Javascript que permite simplificar la manera de interactuar con los documentos HTML, permitiendo manejar eventos,desarrollar animaciones, y agregar interacción con la tecnología AJAX a nuestras páginas web. jQuery esta diseñado para cambiar la forma de escribir código JavaScript. (fuente)
- ✓ JSL: JavaScript Standard Library es un único y pequeño archivo (7.7 KB) con funciones y métodos estándar de JavaScript. Compatible con cualquier navegador que soporte al menos JavaScript 1.2.
- ✓ DHTML Kitchen es un sitio web donde podemos encontrar muchos códigos/script e información sobre DHTML.
- ✓ liberty es una librería básica (simple) para desarrollo web con JavaScript. (fuente)
- ✓ moo.fx es un librería Javascript liviana y pequeña (3KB) con la cual podemos conseguir unos efectos muy interesantes. Trabaja con los frameworks Prototype y Mootools. Simple y fácil de usar. Podemos controlar ó modificar las propiedades CSS y los elementos HTML.
- ✓ overLIB es una librería JavaScript que nos permite mostrar una pequeña caja de información (popup) sobre los enlaces ó link de nuestras páginas web. Brindan asó información a nuestros usuarios sobre a donde nos llevan los links.
- ✓ TurboWidgets son controles JavaScript del lado del cliente para proporcionan un agradable y manejable interfaz de usuario para aplicaciones web estilo AJAX. Construido con Dojo Toolkit, TurboWidgets están diseñados para un uso fácil.
- ✓ overlibmws DHTML Popup Library es una librería DHTML, cuenta con documentación y muchos ejemplos.
- ✓ PlotKit - Javascript Chart Plotting librería en JavaScript para la creación de gráficos. Es soportado por el elemento HTML Canvas, SVG y soporte nativo del navegador. Plokit cuenta con documentación y ejemplos para hacer usarlo en nuestros proyectos sin inconvenientes.
- ✓ qForms JavaScript API es uno de los más completas API JavaScript para la fácil creación y manipulación de formularios en nuestro proyectos web.
- ✓ Zapatec AJAX Suite te brinda una cantidad de herramientas para interfaces de usuarios en tus aplicaciones web, como por ejemplo: calendarios, menús, explorador árbol, formularios, grid, slider, tabs, drag-drgop, efectos y más.
- ✓ Rico es una librería de efectos Ajax disponible en OpenRico que permite simplificar el desarrollo de aplicaciones que utilicen esta tecnología. Mediante Rico es muy sencillo definir la operación básica de Ajax: enviar una solicitud al servidor para que devuelva información. Dispone también de algunos efectos gráficos, tablas actualizables y secciones de drag & drop. (fuente)

- ✓ Sajax es una herramienta de código abierto diseñada para ayudar a los sitios web que usan AJAX framework (también conocido como XMLHttpRequest). Permite al programador llamar a funciones PHP, Perl o Python desde su página web por medio de JavaScript sin necesidad de forzar una actualización de la página en el navegador. (fuente)
- ✓ sardalya herramienta API la creación de páginas DHTML, diseñada para trabajar en todos los navegadores que soportan DOM.
- ✓ script.aculo.us es una librería JavaScript que permite el uso de controles AJAX, drag & drop, y otros efectos visuales en una página web. Se distribuye mediante descargas en varios formatos de archivo, y también está incluido en Ruby on Rails y otros frameworks de desarrollo web.
- ✓ Spry Framework for Ajax es una librería JavaScript de Adobe que facilita el uso de funciones con AJAX. Se encarga de manejar la complejidad interna del AJAX y permite al desarrollador crear facilmente aplicaciones web 2.0.
- ✓ Tacos librería que proporciona componentes AJAX para Tapestry (framework para el desarrollo aplicaciones web en Java). Su funcionalidad está basada en el framework Dojo.
- ✓ TwinHelix nos ofrece proyectos libres DHTML y JavaScript, aunque también XHTML, CSS y CGI.
- ✓ Yahoo! User Interface Library es un paquete de utilidades y controles, escritos en JavaScript, que facilitan la construcción de aplicaciones interactivas (RIA). [Tales como] Drag and drops, animaciones, aplicaciones con Ajax, DOM, etc. Todas muy completas y fáciles de poner en práctica (con pocas líneas de código). La finalidad de esta librería (y de ahí el nombre) es facilitar el desarrollo de aplicaciones ricas del lado del cliente (usuario), logrando elementos visuales e interactivos que incluyen CSS. (fuente)
- ✓ Zebda es una librería en JavaScript para diversos propositos. Se basa en Prototype 1.4.0.
- ✓ Zephyr es un framework para crear aplicaciones AJAX con PHP5. Puedes desarrollar fácilmente aplicaciones empresariales utilizando este robusto framework. Es muy fácil de aprender y muy sencillo de implementar.
- ✓ ZK es un framework Ajax de código abierto que dispone de herramientas ó controles para crear interfaces de usuarios similares a las de escritorio.
- ✓ ext es un framework del lado del cliente para el desarrollo de aplicaciones web. Tiene un sistema dual de licencia: Comercial y Opern Source. Este framework puede correr en cualquier plataforma que pueda procesar POST y devolver datos estructurados (PHP, Java, .NET y algunas otras). (fuente)
- ✓ mootools es un framework JavaScript compacto y modular, orientado a objeto para la creación de aplicaciones web compatible con cualquier navegador.
- ✓ ¿Cónoces de alguna otra librería ó framework para JavaScript, DHTML y AJAX?
  Basado en AJAX, DHTML and JavaScript Libraries.

Anexo II - Selectores CSS que deberíamos conocer

Con la masiva utilización de CSS como sistema de maquetación de páginas web están apareciendo gran cantidad de utilidades y "trucos" para optimizar nuestras hojas de estilos. Esto nos ofrece un mayor control sobre los elementos de nuestro HTML sin necesidad de sobrecargar el HTML con `clases` y `ID's` que realmente no necesitamos.

Para conseguir parte de estas mejoras, deberemos usar los llamados **selectores**, que en resumen son una forma de permitirnos elegir un elemento (o varios) entre todos los que tenemos en nuestro HTML. Similar al funcionamiento de las expresiones regulares para el texto, los selectores nos permiten usar caracteres especiales para referirnos a un elemento o un rango de los mismos.

Selector	Descripción
*	Selector universal, son todos los elementos del CSS
E	`E` representa cualquier elemento del tipo `E` (`span`, `p`, …)
E F	Todos los elementos `F` que sean descendentes de `E`
E > F	Todos los elementos `F` que sean hijos de `E`
E:first-child	De esta forma podemos seleccionar el primer elemento de tipo `E`
E:link , E:visited	Selecciona los elementos `E` que sean un enlace y no hayan sido visitados (`:link`) y los si visitados (`:visited`)
E:active , E:hover , E:focus	Selecciona los elementos de tipo `E` , en sus correspondientes acciones.
E:lang(c)	Cogemos los elementos del tipo `E` que estén en el idioma (humano) especificado en `(c)`.
E + F	Se trata de cualquier elemento `F` inmediatamente después del elemento del tipo `E`
E[foo]	Elementos del tipo `E` con el atributo `foo`
E[foo="ejemplo"]	Elementos del tipo `E` con el atributo `foo` igual a "`ejemplo`"
E[foo~="ejemplo"]	Elementos del tipo `E` con el atributo `foo` contenga "`ejemplo`". Se pueden añadir varias palabras separadas por espacios. (~ =ALT + 0126)
E[lang|="es"]	Similar al anterior, pero se referirá a todos los elemento `E` tal que su atributo `lang` comienze por "`es`". Por ejemplo: "`es_ES`", "`es_CA`",…
E[foo$="ejemplo"]	Elementos del tipo `E` en el que el atributo `foo` termine con "`ejemplo`".
DIV.ejemplo	Todos los elementos `DIV` que sean de la clase `ejemplo`
E#miID	El elemento `E` en el que su `ID` sea igual `miID`

Ampliando los selectores

Además de estos no pocos selectores podemos, aún más, conseguir filtrar la búsqueda de elementos, para ello usaremos pseudo-elementos.

`:first-line`

Se refiere a la primera línea del elemento, normalmente usado para elementos de texto.

```
p {font-size: 12pt}
p:first-line {color: #0000FF; font-variant: small-caps}

<p>Some text that ends up on two or more lines</p>
```

Propiedades:

- ✓ font properties
- ✓ color properties
- ✓ background properties
- ✓ word-spacing
- ✓ letter-spacing
- ✓ text-decoration

- ✓ vertical-align
- ✓ text-transform
- ✓ line-height
- ✓ clear

:first-letter

La primera letra del elemento, también suele usarse para elementos de texto.

```
p {font-size: 12pt}
p:first-letter {font-size: 200%; float: left}

<p>The first words of an article.</p>
```

Propiedades:

- ✓ font properties
- ✓ color properties
- ✓ background properties
- ✓ margin properties
- ✓ padding properties
- ✓ border properties
- ✓ text-decoration
- ✓ vertical-align (only if 'float' is 'none')
- ✓ text-transform
- ✓ line-height
- ✓ float
- ✓ clear

:before

Elemento usado para insertar algún contenido delante de un elemento.

```
h1:before { content: url(beep.wav) }
```

:after

Elemento usado para insertar algún contenido al final del elemento.

```
h1:after { content: url(beep.wav) }
```

pseudo-elementos y CSS clases.

Nos permite encadenar varios selectores para conseguir focalizar en un elemento un pseudo-elemento concreto.

```
p.article:first-letter {color: #FF0000}
<p class="article">A paragraph in an article</p>
```

Multiples pseudo-elementos

Además nos permite utilizar varios pseudo-elementos sobre un mismo elemento.

```
p {font-size: 12pt}
p:first-letter {color: #FF0000; font-size: 200%}
p:first-line {color: #0000FF}

<p>The first words of an article</p>
```

Compatibilidad

Pseudo-elemento	IE	F	N	W3C
:first-letter	5	1	8	1
:first-line	5	1	8	1
:before		1.5	8	2
:after		1.5	8	2

Anexo III - 10 extensiones de firefox para el desarrollo web

Sin lugar a dudas, una de las mejores cosas de **Firefox** es la posibilidad de agregar **extensiones**. Aquí va un listado de 10 extensiones bastante útiles para cuando estás trabajando en tu blog u otro sitio web:

1. **Web Developer**, probablemente el más clásico de todos. Una barra con un montón de funcionalidades, desde proporcionar información sobre elementos, herramientas de validación, reglas y guías, ajustar el tamaño de la ventana, delinear elementos (¡muy práctico!), etc.
2. **Firebug**, el próximo gran clásico. Si bien es cierto que repite algunas de las funcionalidades del anterior, también lo es que agrega muchas más, en especial sus herramientas para trabajar con JavaScript, **DOM**, actividad de la red, etc.
3. **YSlow**, un complemento para el complemento anterior. YSlow analiza una página de acuerdo a las directrices utilizadas por **Yahoo** para sitios de alto rendimiento y entrega un detallado reporte junto a algunas herramientas. Ideal para detectar y prevenir "cuellos de botella" en la carga de tu web.
4. **Screengrab!**, simplemente, la mejor herramienta que he probado para crear y guardar *screenshots* de una ventana, la porción visible o toda la página.
5. **IE Tab**, integra el motor de renderizado de **MSIE** en Firefox, con sólo un click es posible cambiar entre **Gecko** e Internet Explorer.
6. **Dust-Me Selectors**. La versión corta: una extensión para encontrar selectores CSS que no se utilizan. La versión larga: extrae todos los selectores de las hojas de estilo de la página que se está viendo y luego analiza la página para comparar qué selectores no se están utilizando.
7. **ShowIP**, como su nombre lo indica, muestra la dirección **IP** y permite consultar algunos servicios por IP o nombre del dominio, como **whois**, **Netcraft**, **traceroute** o localización geográfica.
8. **Make Link**, agrega una entrada al menú desplegado con el botón secundario del mouse para crear fácilmente enlaces en formato **HTML**, BBCode (para foros) o como simple texto. Además es posible crear nuestras propias combinaciones en base a algunas variables, con lo que las posibilidades son infinitas.
9. **FireFTP**, todo un cliente de **FTP** dentro de Firefox. Especialmente indicado para instalaciones **portables**.
10. **Mouse Gestures**, la única extensión de "uso general" que se me ha hecho verdaderamente indispensable. Permite ejecutar comandos comunes con simples gestos de ratón; recomiendo especialmente habilitar los gestos *rockers* con la rueda.

Anxo IV - Efectos con jQuery

Category: Effects

The jQuery library provides several techniques for adding animation to a web page. These include simple, standard animations that are frequently used, and the ability to craft sophisticated custom effects.

.animate()
Perform a custom animation of a set of CSS properties.
.clearQueue()
Remove from the queue all items that have not yet been run.
.delay()
Set a timer to delay execution of subsequent items in the queue.
.dequeue()
Execute the next function on the queue for the matched elements.
.fadeIn()
Display the matched elements by fading them to opaque.
.fadeOut()
Hide the matched elements by fading them to transparent.
.fadeTo()
Adjust the opacity of the matched elements.
.fadeToggle()
Display or hide the matched elements by animating their opacity.
.finish()
Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.
.hide()
Hide the matched elements.
jQuery.fx.interval
The rate (in milliseconds) at which animations fire.
jQuery.fx.off
Globally disable all animations.
.queue()
Show or manipulate the queue of functions to be executed on the matched elements.
.show()
Display the matched elements.
.slideDown()
Display the matched elements with a sliding motion.
.slideToggle()
Display or hide the matched elements with a sliding motion.
.slideUp()
Hide the matched elements with a sliding motion.
.stop()
Stop the currently-running animation on the matched elements.
.toggle()
Display or hide the matched elements.